ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN ANDROID PARA FACILITAR LA COMUNICACIÓN PARA PERSONAS CON DISCAPACIDAD AUDITIVA.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN INGENIERÍA ELECTRÓNICA Y REDES DE INFORMACIÓN

JONATHAN ANDRÉS AYMACAÑA GUERRÓN

DIRECTOR: XAVIER ALEXANDER CALDERÓN HINOJOSA

Quito, septiembre 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Jonathan Andrés Aymacaña Guerrón, bajo mi supervisión.

XAVIER ALEXANDER CALDERÓN HINOJOSA
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo Jonathan Andrés Aymacaña Guerrón, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Jonathan Andrés Aymacaña Guerrón

DEDICATORIA

A Dios, Jesús y el Espíritu Santo que me han cuidado durante todo el camino llamado vida.

Este trabajo se lo dedico en su totalidad a mi padre y mi hermano que me brindan su apoyo constante y me hicieron conseguir este objetivo. Por los consejos, enseñanza de superación, por enseñarme valores buenos que han forjado una persona de bien. Todos los logros del pasado y del futuro serán gracias a ustedes.

Andrés Aymacaña

AGRADECIMIENTO

Agradezco de manera especial a mi padre que es y será la guía y admiración de ser humano y esfuerzo.

Agradezco a mi hermano con quien hemos pasado muchas adversidades, pero siempre encontré un apoyo pese a estas.

Agradezco a mis abuelos Víctor y Lucila que nunca dejaron que caiga y salga de mi camino.

Agradezco a mi novia Jessica Elizabeth quien me dio fuerza y amor para acabar la tesis sin dejar el trabajo y además me llena todos los días de amor.

Agradezco a mi madre que me dio la vida.

Agradezco a mis primos Oscar, Brandon, Pico, Johan y Balta quienes me cuidaron en mis locuras y me permitieron brindarles mi confianza.

Agradezco a mis tías Maribel, Jenny y Maritza que son un ejemplo de mujeres.

Agradezco a todos los amigos y amigas de GBTheory en especial a Cesar, Sandy, Paul, Toño, Alex, Jorge, Carlos y Ángel mi Sensei del lol.

Agradezco a mi director de trabajo de titulación quien es una pieza fundamental para mi presente trabajo laboral gracias a enseñarme SNMP y mi trabajo de titulación, muchas gracias por dedicar su tiempo, confianza y esfuerzo para lograr terminar con este ciclo de mi vida.

Andrés Aymacaña

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente proyecto de titulación tiene como objetivo el desarrollo de un prototipo de aplicación Android, para facilitar la comunicación para personas con discapacidad auditiva.

Se utilizó la metodología de desarrollo ágil iterativa “Scrum” para el desarrollo de este proyecto porque se basa en procesos que incrementan y cambian a medida que se avanza. Este documento está dividido en cuatro capítulos que se detallan a continuación.

En el primer capítulo se realiza una introducción, se describe el objetivo general, objetivos específicos, alcance y herramientas a utilizar para el desarrollo del proyecto.

En el segundo capítulo se detalla el diseño de las interfaces, el desarrollo de la aplicación y la implementación del proyecto.

En el tercer capítulo se indica las pruebas de funcionalidad y análisis de resultados.

En el cuarto capítulo se detalla las conclusiones y recomendaciones obtenidas en el desarrollo del proyecto de titulación.

Finalmente, se incluye como anexos guías para los diferentes roles, encuestas, el código del proyecto completo y la guía para poder modificar y compilar el código en cualquier computador con Android Studio.

Palabras clave: Scrum, JAVA, transcripción, discapacidad auditiva, android.speech.RecognizerIntent, java.lang.Object, tiempo real, firebase, autenticación, Android.

ABSTRACT

The objective of this project is to develop an Android application prototype, to facilitate communication of people with hearing disabilities.

The iterative agile development methodology "Scrum" was used for the development of this project, It is based on processes that increase and change as it progresses. This document is divided into four chapters.

First chapter shows the general objective, specific objectives, scope and tools to be used for the development of the project are described.

Second chapter details the design of the interfaces, the development of the application and the implementation of the project.

Third chapter indicates the functionality tests and analysis of results.

Fourth chapter details the conclusions and recommendations obtained after in the development of the degree project.

Finally, guides for the different roles, surveys, the complete project code and the guide to be able to modify and compile the code on any computer with Android Studio are included as annexes.

Keywords: Scrum, JAVA, transcription, hearing impaired, android.speech.RecognizerIntent, java.lang.Object, realtime, firebase, authentication, Android.

1. INTRODUCCIÓN

En la actualidad, uno de los problemas que tienen las personas con discapacidad auditiva es que no cuentan con una aplicación móvil que integre características de varias aplicaciones, la cual permita la comunicación de manera fácil con otras personas. Es por tal motivo que estas personas tienen mayor dificultad para superarse dentro del ámbito laboral y personal, ya que al tener como impedimento la comunicación con otras personas tiene como consecuencias el no pueden acudir a cursos, eventos, conferencias, entre otros.

Debido a esto se plantea como una alternativa para solucionar este problema el desarrollo de una aplicación móvil para el Sistema Operativo Android, que integre características de varias aplicaciones como mensajería, sencillez de uso, posibilidad de configuración, enfoque directo a trabajar en smartphone con sistema operativo Android, seguridad, ubicación todo esto para que facilite la comunicación de personas con discapacidad auditiva.

En caso de que no se desarrolle esta aplicación las personas con discapacidad auditiva no dispondrán de una aplicación móvil que abarque las características adecuadas para facilitar su comunicación y tendrán que comunicarse a través del lenguaje de señas, leer los labios o disponer de varias aplicaciones con varias cuentas instaladas en su dispositivo móvil.

1.1. OBJETIVOS

Desarrollar un prototipo de aplicación Android para facilitar la comunicación para personas con discapacidad auditiva.

Los objetivos específicos de este proyecto de titulación son:

- Analizar los conceptos fundamentales para el desarrollo del proyecto propuesto.
- Diseñar los módulos del sistema considerando los requisitos que debe cumplir el prototipo.
- Implementar los módulos del sistema según el diseño realizado.
- Analizar los resultados obtenidos en las pruebas realizadas.

1.2. ALCANCE

Se pretende crear una aplicación Android la cual facilite la comunicación para personas con discapacidad auditiva.

Se prevé que la aplicación desarrollada transcriba de audio a texto con ayuda de android.speech.RecognizerIntent que extiende de java.lang.Object la cual pertenece a Android Studio y cuenta con más de 30 funciones, objetos y métodos que brinda escritura,

transcripción, seguridad, entre otras, además trabaja en conjunto con herramientas de Google [1]; también permite la opción de guardar o descartar la transcripción, brinde información acerca de todas las transcripciones que el usuario guardó previamente, se pueda mostrar la ubicación actual al familiar asignado y además enviar la ubicación a un contacto dentro de su aplicación WhatsApp, el lenguaje de la conversación será en inglés o español, el usuario podrá registrarse con solo tener un correo electrónico y crear una clave para luego autenticarse desde cualquier dispositivo móvil con un sistema operativo Android [2], se puede enviar mensajes escritos dentro de la aplicación al familiar asignado, se puede guardar informes de como utilizó la aplicación y solo se necesita una conexión a internet.

La información respecto al usuario y las varias conversaciones serán almacenadas en una base de datos [3] y en el momento que se necesite la información dentro de la aplicación será obtenida, procesada y será reflejada en la pantalla del dispositivo dependiendo del tipo de usuario.

La arquitectura del sistema se muestra en la Figura 1.1

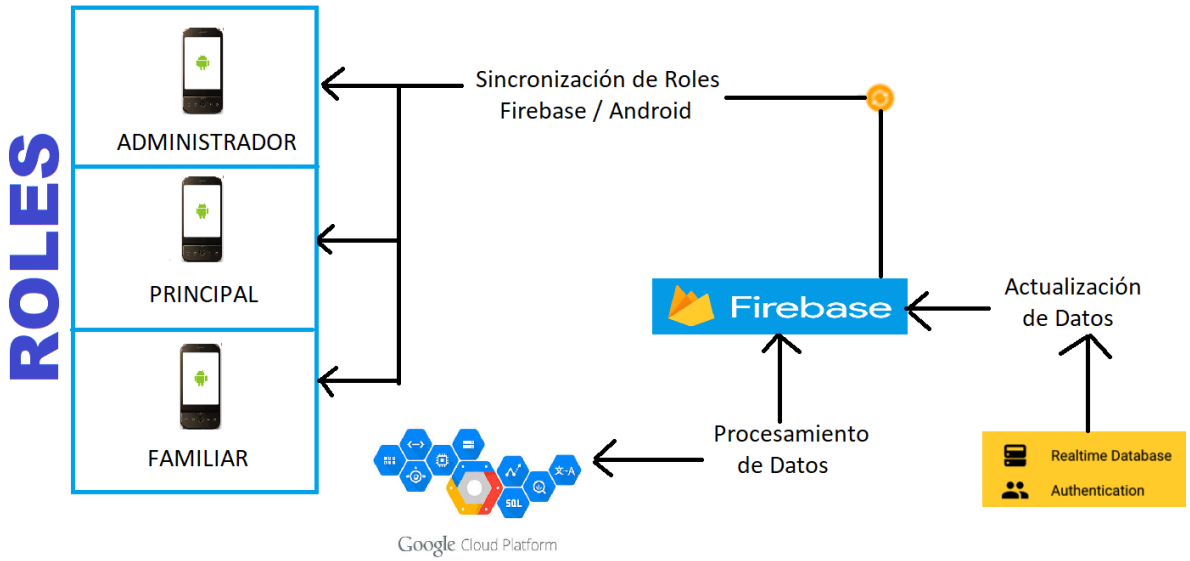


Figura 1.1. Arquitectura del sistema

El desarrollo del prototipo se basará en la metodología de desarrollo ágil iterativa “Scrum”.

Para la base de datos se utilizará Firebase de Google la cual es una base de datos no relacional donde se alojará toda la información a ser gestionada y será recuperada en el momento en el que el usuario la requiera para su uso, es decir, descripción del usuario, ubicación, transcripciones entre otros se actualizarán constantemente.

Se tendrá tres tipos de roles y cada uno realizará las siguientes actividades:

- **Administrador:** Es el encargado de ver detalles generales de cada usuario, además podrá ver el informe de utilización de cada usuario y un informe conjunto de cómo se está usando la aplicación, no podrá ver transcripciones, mensajes ni ubicación de los usuarios.

- **Principal:** Es el rol primordial de la aplicación, es quien inicia la transcripción y podrá guardar cada una según el considere, o eliminar las previamente guardadas, podrá mostrar la ubicación actual a su familiar asignado y además enviar la ubicación a un contacto dentro de su aplicación WhatsApp, dependiendo como utilice la aplicación podrá ver un informe o un reporte propio, si desea comunicarse con su familiar a través de texto tendrá una opción de envío y recepción de mensajes.

- **Familiar:** Es el encargado de visualizar a detalle la ubicación y el recorrido del usuario Principal si este lo permite, podrá ver un informe de como utiliza la aplicación el usuario Principal al que esté asignado y podrá enviar y recibir mensajes.

Cada rol tendrá una interfaz diferente.

El prototipo contará con al menos los siguientes módulos:

- **Registro:** Los usuarios se registrarán al rol que pertenezcan con sus datos, entre ellos se incluirán un correo y una contraseña para autenticarse en la aplicación. El correo debe ser creado por el usuario previamente y para la contraseña se utilizará la parte de Authentication propia de Firebase donde se puede configurar los parámetros de hash de contraseña y en conjunto con Android Studio asegurar que se cumplan estos parámetros por medio de programación.

- **Inicio de sesión:** Los usuarios podrán iniciar sesión mediante el ingreso del correo y su contraseña previamente registrados, luego se visualizará la funcionalidad de la aplicación de acuerdo con el tipo de rol. Cabe recalcar que para comprobar la contraseña y el correo ingresados se tiene que trabajar en conjunto la parte de interfaz gráfica del usuario, el objeto por parte de Firebase integrado en la aplicación, Authentication de Firebase y Realtime Database de Firebase.

- **Recuperar Contraseña:** Módulo en el que un usuario de cualquier rol puede recuperar su contraseña en caso de que se haya olvidado o quiera cambiarla, por lo tanto se debe comprobar que existe el correo ingresado para ello utilizamos el objeto por parte de Firebase integrado en la aplicación, Authentication de Firebase y Realtime Database de Firebase; más tarde se debe confirmar el cambio de clave por lo que se utiliza Authentication de Firebase y herramientas propias que integra Sign-in method y configuración de Templates para crear un enlace, enviar al correo y la interacción con el usuario.

- **Ver Ubicación:** Módulo donde el usuario principal y familiar podrán observar la ubicación actual del usuario principal, además se podrá enviar la ubicación a un contacto de WhatsApp. Este módulo hará uso de la API de Google Maps perteneciente a Google Cloud Platform.

- **Reporte:** Módulo en el cual un usuario podrá ver un informe individual de la utilización de la aplicación, además ver tema y fecha de todas las transcripciones guardadas hasta ese momento. Para esto se utilizará una búsqueda en Realtime Database de Firebase a diferentes nodos dentro de la estructura, tomando en cuenta el usuario integrado en la aplicación, se captura información necesaria y se la procesa dentro de Android Studio generando un documento que se puede ver tanto en la aplicación como guardarlo en su smartphone.

- **Transcripción:** Módulo donde el usuario perteneciente al rol principal podrá transcribir sus conversaciones a texto para luego guardarlas en la base de datos si así lo considera, podrá descartar conversaciones, pausarlas o continuar, también tendrá opción de aumentar el tamaño de letra, la transcripción reconocerá automáticamente si el idioma es español o inglés. Para esto se utiliza android.speech.RecognizerIntent que extiende de java.lang.Object la cual pertenece a Android Studio y trabaja en conjunto con herramientas de Google.

- **Portafolio:** Módulo donde se podrá ver el contenido de las transcripciones guardadas con anterioridad, también se podrá eliminar o editar cualquier transcripción. Para esto se utilizará una búsqueda en Realtime Database de Firebase a diferentes nodos dentro de la estructura, tomando en cuenta el usuario integrado en la aplicación, se captura información necesaria y se la procesa dentro de Android Studio.

- **Mensajes:** Módulo donde se podrá comunicar a través de texto entre un usuario del rol Principal con un Familiar utilizando la propia aplicación. Para esto tendremos una sincronía entre la memoria del smartphone y Realtime Database ya que guardaremos los mensajes en Realtime Database de Firebase lo enviaremos de un nodo a otro y a continuación se mostrará al receptor de manera visual.

- **Menú:** Cada usuario podrá ver un menú diferente ordenado en función de cada rol.

Este trabajo de titulación tiene un producto final demostrable.

1.3. MARCO TEÓRICO

1.3.1. METODOLOGÍA DE DESARROLLO SCRUM

La metodología de desarrollo ágil SCRUM es un framework que se utiliza en trabajo de equipo autodirigido cooperativo para proyectos complicados donde se necesita supervisar la evolución del producto, aplicando buenas prácticas y entregas parciales. [4]

1.3.1.1. Principales características de la Metodología Scrum

A continuación, se dicta características que brinda el aplicar la metodología Scrum a un proyecto:

- Tiene un equipo de trabajo autodirigido.
- Utiliza un conjunto de reglas para administrar el proyecto de forma ágil.
- No dispone de prácticas especificas utilizadas en ingeniería.
- Los ítems de la lista Product Backlog son los requerimientos de nuestro proyecto.
- Se construye una serie de *Sprints* para obtener el producto final.

1.3.1.2. Herramientas y Prácticas

Product Backlog List

Es una lista de requerimientos priorizados que ayuda a saber que se realizará en el proyecto y en qué orden. Al iniciar un proyecto es muy complicado obtener todos los requerimientos que se implementarán en el proyecto, pero casi siempre se obtienen los más importantes para iniciar un *Sprint*. Esta lista puede modificarse a medida que se obtiene nueva información sobre el producto por parte del cliente, pero con una restricción que solo puede cambiar entre secuencias de *Sprints*, con el objetivo de hacer un producto final útil, competitivo y correcto. El Product Owner es el único encargado de agregar y modificar la lista. [4]

Sprints

Es un procedimiento donde se adapta ideas como requerimientos, recursos, conocimientos y programación de tiempo. Se debe tomar en cuenta que durante un Sprint el producto debe ser diseñado, codificado y probado, estos ciclos sirven para desarrollar o mejorar la funcionalidad y obtener nuevos incrementos. Un *Sprint* debe tener una planificación en tiempo de 1 a 4 semanas, no se puede introducir cambios dentro de un mismo Sprint y se puede incluir también restricciones que afectarían al proyecto final. [4]

Sprint Backlog

Es el punto de referencia a entrada de un *Sprint*, al igual que Product Backlog List es una lista, pero de requerimientos que van a ser implementados en el siguiente Sprint, este Sprint Backlog es seleccionado por el Scrum Team, Scrum Master y Product Owner una vez priorizados los requerimientos. [4]

Stabilization Sprints

Son *Sprints* que se enfocan en encontrar fallas o defectos, no en crear funcionalidades para el proyecto. Son útiles cuando se preparan pruebas beta en el producto o este no llena los límites que se esperaba. [4]

1.3.1.3. Fases de Scrum

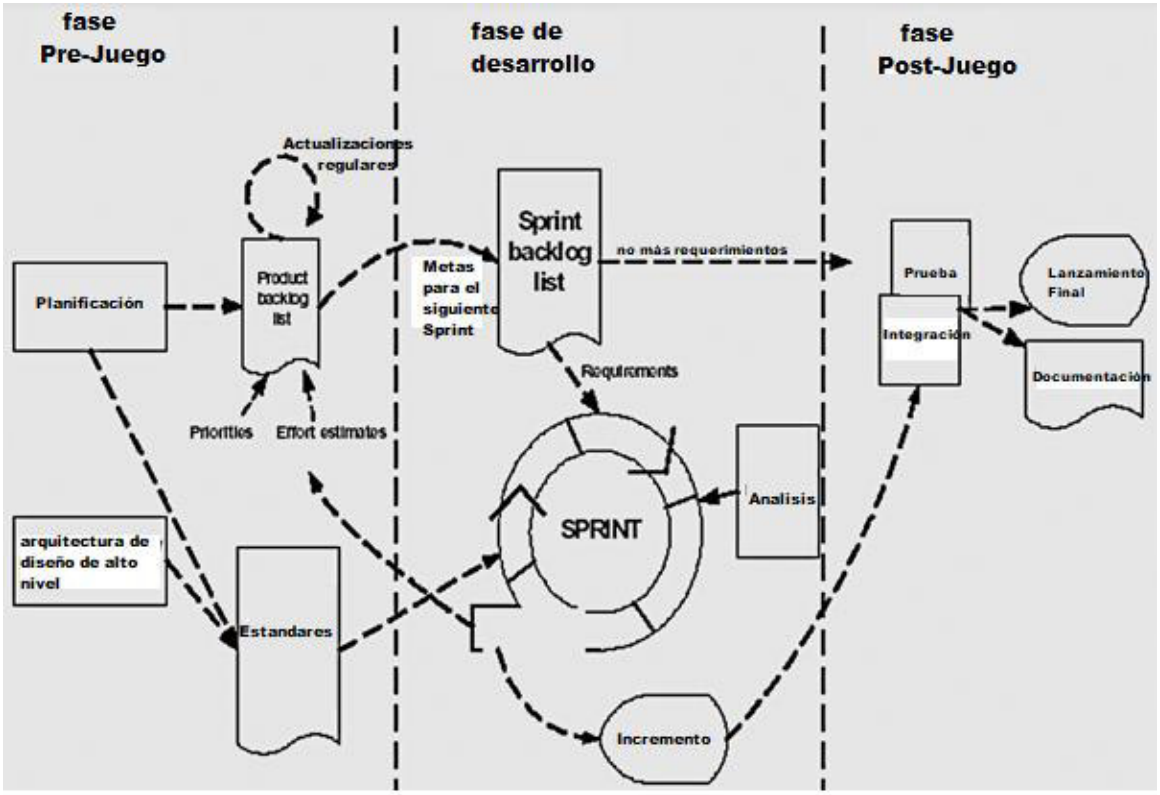


Figura 1.2. Fases de Scrum

Fuente: [4]

Pregame (Planificación o Pre-juego)

Esta fase consta de dos subfases:

- **Planning (Planeación)**

  Indica la definición del sistema que se pretende construir, es decir la idea general que permitirá crear la lista Product Backlog. [4] [5]

- **Architecture/High level Design (Arquitectura de diseño de alto nivel)**

  Es la planificación de los elementos de la Product Backlog List, a modo de ejemplo se puede asumir que se desea implementar una mejora para un sistema, entonces debe identificar los cambios que se necesita, pero tomando en cuenta el impacto que se puede tener y se agrega a la lista Product Backlog. [4] [5]

Development (Fase de Juego)

Es conocida también como Game Phase y es la parte ágil de toda la metodología Scrum. Es la fase donde se espera caos, donde se encuentra variables impredecibles en el tiempo, requerimientos, tecnologías, recursos y herramientas para la implementación. Scrum propone que no se realice esfuerzo en estas variables al comenzar el proyecto sino controlarlas, adaptarse y crear cambios de manera flexible. [4] [5]

1.3.1.4. Roles y Responsabilidades

Scrum Master

Es un rol asignado al Líder o Gerente del proyecto, se enfoca en supervisar que el proyecto se lleve a cabo cumpliendo las diferentes prácticas y reglas que brinda Scrum. Su principal cargo es remover los impedimentos y minimizar los riesgos.

Product Owner

Es el rol encargado de la responsabilidad de comunicar, controlar y administrar el Backlog List, tiene una visión clara del proyecto y se lo conoce también como el Product Manager. [4] [5]

Scrum Team

Es el equipo del proyecto entre 5 y 10 personas, este puede organizarse de diferente forma para cumplir los objetivos del *Sprint* en tareas como:

- Effort Estimation.
- Crear el Sprint Backlog.
- Revisar la Product Backlog List.
- Sugerir obstáculos para ser removidos.

Customer

Es el cliente y participa en ideas que se convertirán en tareas. [4] [5]

Management

Es el encargado de tomar decisiones finales para reducir la lista Product Backlog y seleccionar objetivos que llevaran a concluir el proyecto. [4]

1.3.1.5. Fundamentación

Algunas razones para utilizar la Metodología de Desarrollo Scrum son:

- El proyecto depende de un sistema modular que permite desarrollar una base funcional mínima e ir aumentando, modificando y comprobando funcionamientos anteriores y actuales.
- Crea entregas en lapsos de tiempo para acoplarse a las ideas del cliente.
- Puede incrementar funcionalidades indicadas al inicio del proyecto.
- Se puede variar el orden de trabajo si se encuentra un inconveniente o dependencias.

1.3.2. SISTEMA OPERATIVO ANDROID

El sistema operativo Android es utilizado en 2500 millones de dispositivos desde equipos pequeños físicamente hasta ordenadores potentes. Está basado en código abierto por lo que está disponible a nivel mundial tanto para diseñadores, desarrolladores como fabricantes de dispositivos. [6]

Este sistema operativo móvil está basado en Linux, lo que permite a los usuarios poder crear código con mínimas restricciones y ampliar la funcionalidad de dispositivos que cuentan con este sistema operativo.

1.3.2.1. Versiones de Android

Android fue lanzado en el año 2008 con su versión 1.0 y a medida que pasa el tiempo han surgido varias actualizaciones hasta llegar a su versión comercial de Android 11, en cada actualización se corrigen errores del programa y se implementa nuevas funciones. [7]

A continuación, se nombra algunas versiones de Android en las cuales se corrió este proyecto.

1.3.2.1.1. Android 6.0 Marshmallow

Figura 1.3. Android 6.0 Marshmallow

Marshmallow (Figura 1.3) se lanzó el 28 de mayo de 2015 tiene como última versión estable la 6.0.1 (MOI10E) lanzada el 3 de octubre de 2017; se detalla a continuación características relevantes [8]:

- Acceso directo más inteligente.
- Batería más eficiente e inteligente para su ahorro.
- Gestión de permisos acorde al usuario.
- Seguridad simplificada con huella dactilar.

1.3.2.1.2. Android 9 pie

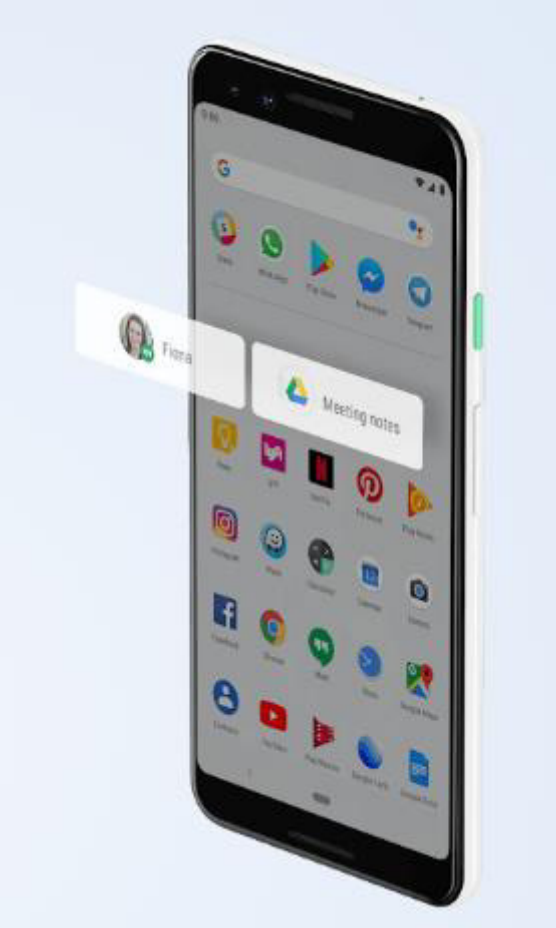


Figura 1.4. Android 9 pie [9]

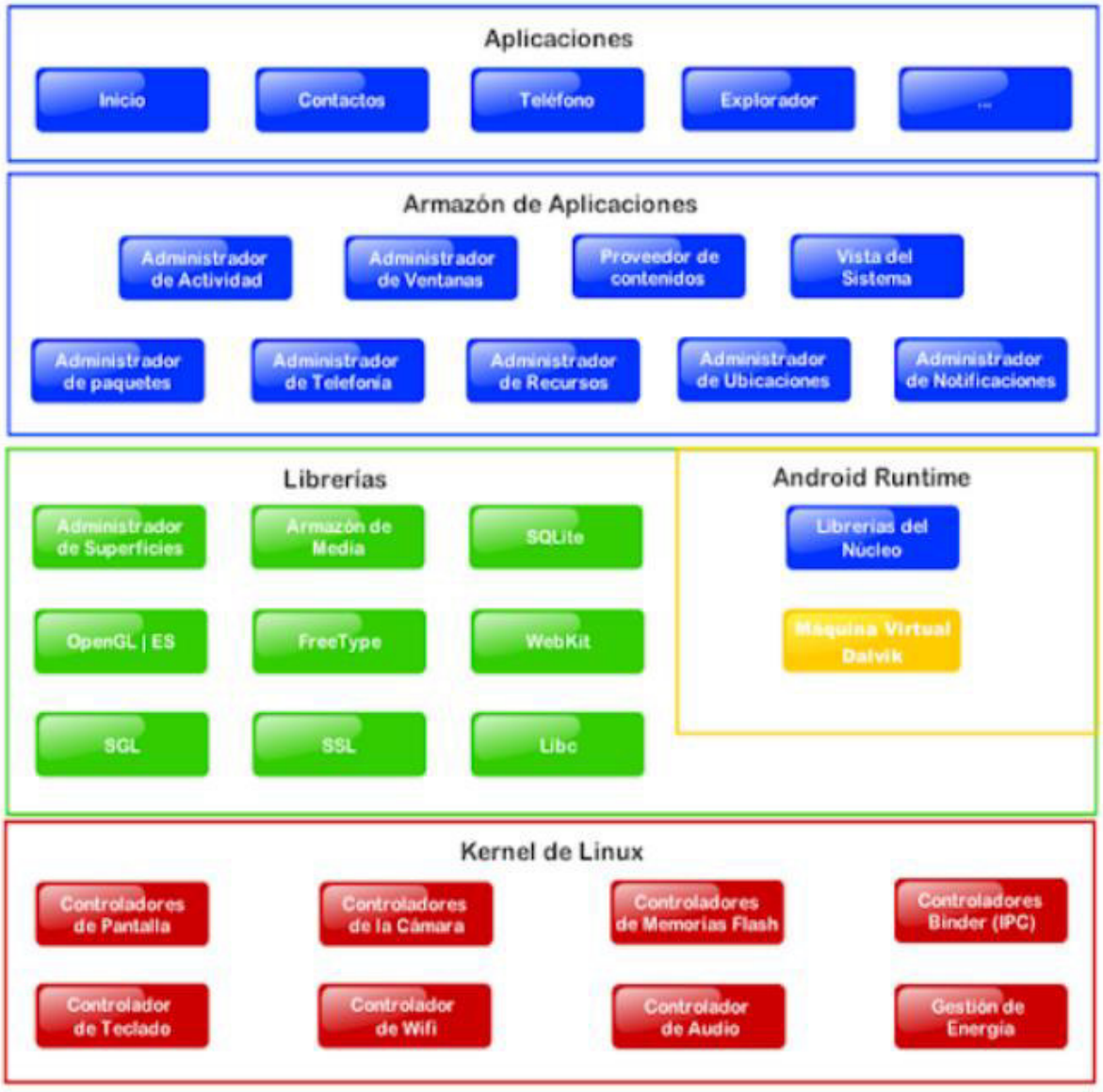
Pie (Figura 1.4) se lanzó el 6 de agosto de 2018 como su última versión; se detalla a continuación características relevantes [9]:

- Batería adaptiva, selecciona las aplicaciones que utiliza el usuario y les da prioridad.
- Brillo adaptivo, se ajusta automáticamente.
- Cambio entre aplicaciones mediante gestos.

- Información de donde el usuario consume mayor tiempo.

1.3.2.2. Arquitectura Android

La arquitectura Android contiene una pila de software como se indica en la Figura 1.5 la que contiene sistema operativo (kernel de Linux), middleware (librerías y Android runtime) y aplicaciones para usuario (aplicaciones y armazón de aplicaciones).



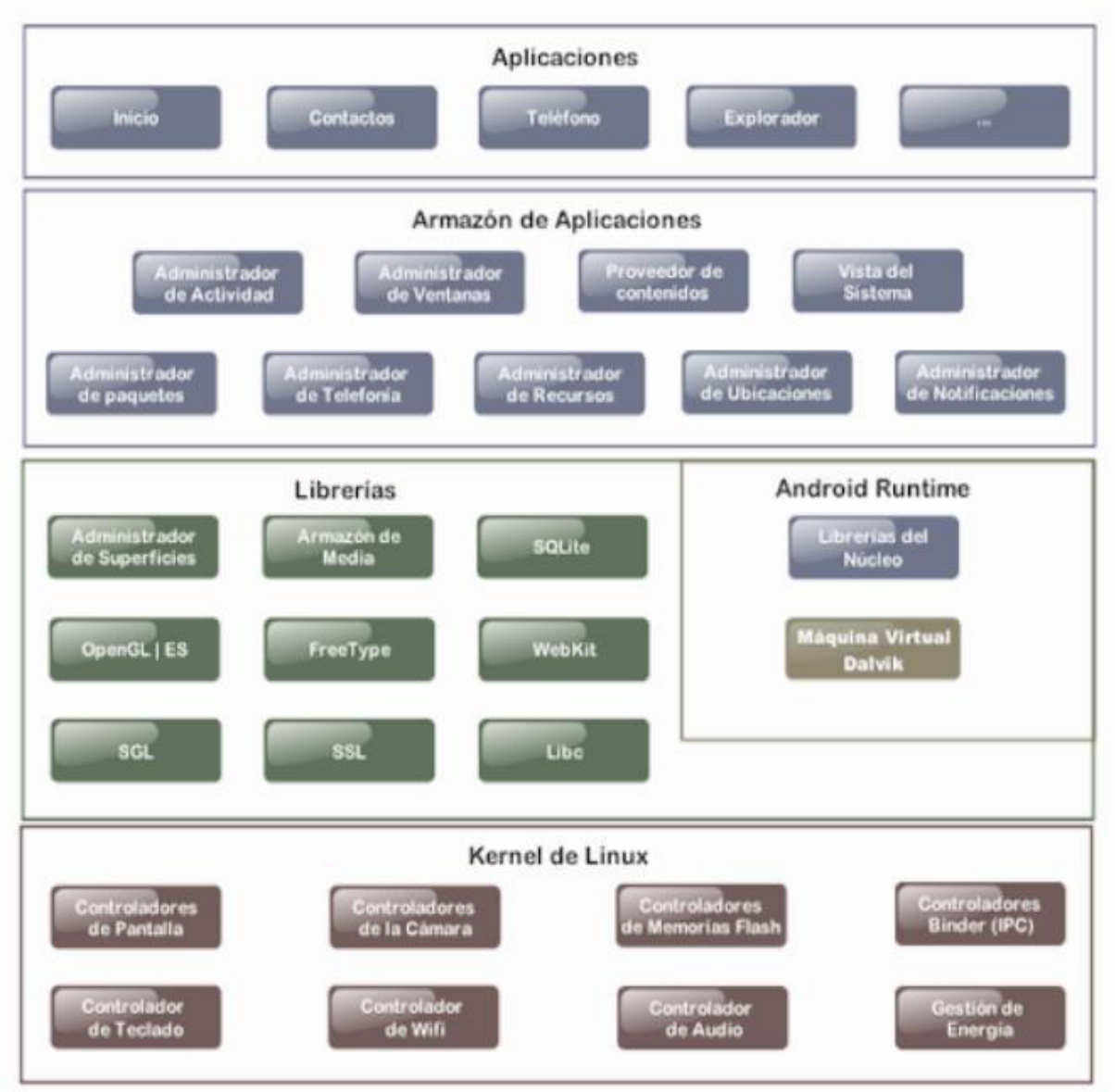


Figura 1.5. Arquitectura de Android [10]

Aplicaciones

En este nivel se alojan todas las aplicaciones que contiene Android por defecto y también las creadas por el usuario o que se vayan añadiendo posteriormente independientemente si son de terceras entidades. Todas las aplicaciones obligadamente utilizan servicios, APIs y librerías de niveles anteriores.

Framework de Aplicaciones

Es un conjunto de herramientas de desarrollo, es decir, se utiliza el mismo conjunto de APIs y de framework para toda aplicación desarrollada en Android sin importar si los creadores son terceros. Algunas de las APIs más importantes son:

- Activity Manager.

- Windows Manager.
- View System.
- Location Manager.

Librerías

En esta capa se aloja las librerías utilizadas para varias tareas por Android, son escritas en lenguaje de programación C/C++ y es una de las partes con mayor importancia en la arquitectura Android.

Android Runtime

Está ubicada en el entorno de ejecución al mismo nivel que las librerías.

Núcleo o Kernel de Linux

Se utiliza como una capa de abstracción para el hardware del dispositivo, contiene los drivers necesarios para realizar llamadas a cualquier componente hardware.

1.3.2.3. Android Studio

Es un entorno de desarrollo integrado diseñado para crear Apps para Android, tiene un sistema basado en Gradle[1], emulador rápido, entorno donde se desarrolla para todos los dispositivos Android, ejecución de app sin reiniciarla, integración con GitHub, compatibilidad con Google Cloud Plataform, entre otras. [11]

Android Studio (Figura 1.6) está disponible para Windows 7,8 y 10, macOS, GNU/Linux, entre otros, algunas características básicas para un correcto funcionamiento previo a su instalación son:

- 4 GB de RAM.
- 4 GB espacio en disco.
- Versión de Java JDK instalada.

[1] **Gradle:** Sirve para automatizar la construcción de nuestro código de manera flexible utilizando varios lenguajes de programación incluido Java. Obtenido de: https://www.arquitecturajava.com/que-es-gradle/

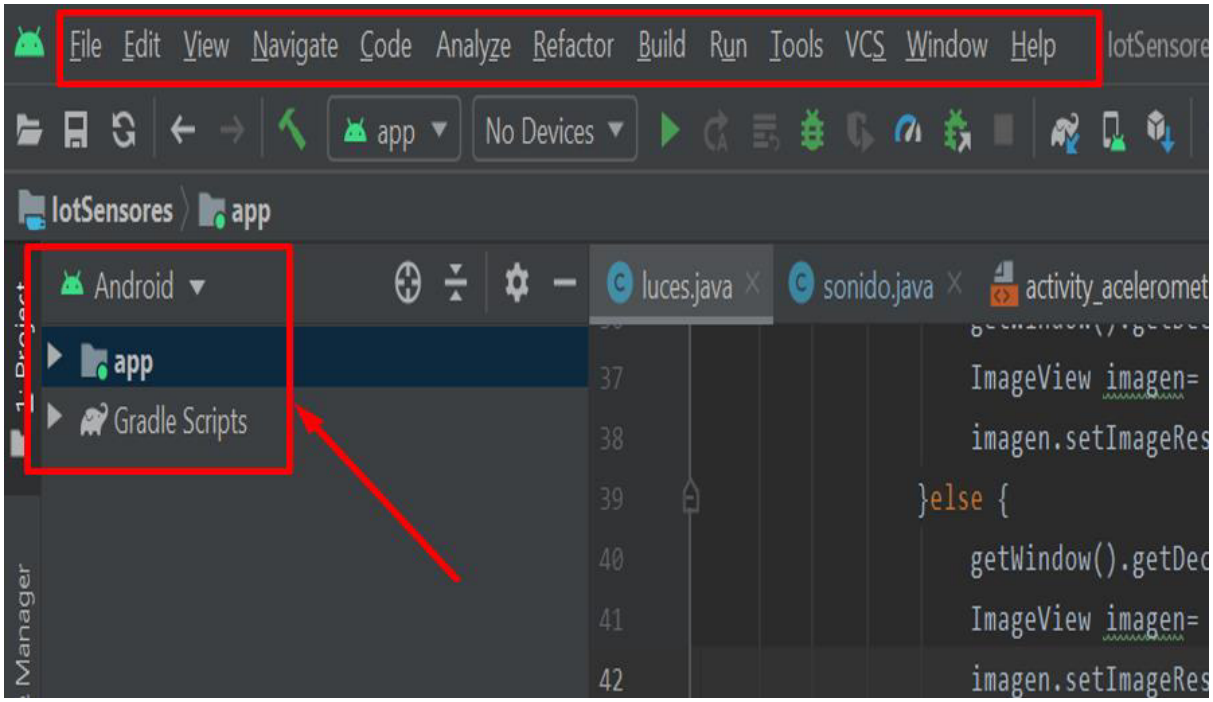


Figura 1.6. Arquitectura de Android

1.3.3. JAVA

Es un lenguaje orientado a objetos, es decir se tiene una idea básica a la forma humana de pensar las cosas, es un lenguaje multiplataforma, es seguro ya que la máquina virtual al ejecutar el código comprueba seguridad y el propio lenguaje tiene características seguras.

Para poder trabajar con Java es necesario tener la herramienta proporcionada por Sun el creador de java, que es el Java Development Kit (JDK). [12]

1.3.4. FIREBASE

Firebase es una creación de Google específicamente diseñada para el complemento en desarrollo de aplicaciones móviles, cuenta con varias funciones [13] :

- Base de datos.
- Informes.
- Mensajería.
- Autenticación.

Gracias a las diferentes APIs de Google es posible tener a Firebase en distintos sistemas operativos como Android y iOS.

1.3.4.1. Firebase RealTime Database

Como se indica en la Figura 1.7. es una base de datos NoSQL que está diseñada para el desarrollo en la nube que permite sincronizar y almacenar datos en tiempo real[2] utilizando formato JSON. Estos datos se mantienen disponibles utilizando la infraestructura de Google con características como SDK[3] de iOS, Android y Web, consultas, replicaciones y facilidad. [14]

Figura 1.7. Realtime Database

1.3.4.2. Firebase Authentication

Como se muestra en la Figura 1.8.; es diseñada para el desarrollo en la nube que permite facilitar la creación de sistemas de autenticación de manera segura, cuenta con autenticación de extremo a extremo y puede utilizar correo electrónico y contraseña. [15]

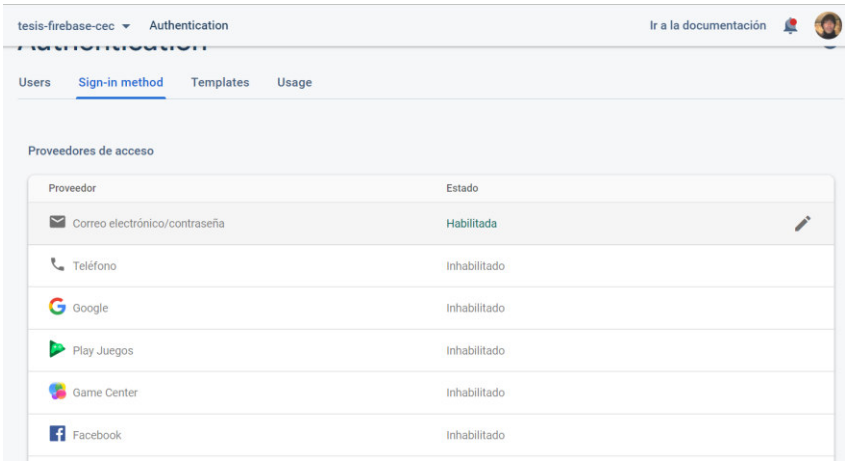


Figura 1.8. Opciones de autenticación

[2] **Tiempo Real:** Actualización y sincronización de datos se los realiza muy rápidamente en cuestión de milisegundos. Obtenido de: https://firebase.google.com/products/realtime-database/?hl=es-419

[3] **SDK:** Software Development Kit, es un grupo de herramientas diseñado para programar aplicaciones móviles. Obtenido de: https://www.atinternet.com/es/glosario/sdk/

1.3.5. API DE GOOGLE

Application Programming Interface o conocida como API, es código que ayuda a la comunicación e integración de aplicaciones independientes con varios servicios ya creados por Google. Las funciones que contiene permiten ser reutilizadas por los desarrolladores evitando volver a escribir funciones con la certeza que funcionarán.

La única necesidad que se tiene es la creación de una cuenta de desarrollador en Google. Cuando una aplicación necesita utilizar un servicio se genera una solicitud HTTPS que permite la comunicación entre la aplicación y Google siendo totalmente transparente para el usuario final. [16]

1.3.5.1. Maps SDK para Android

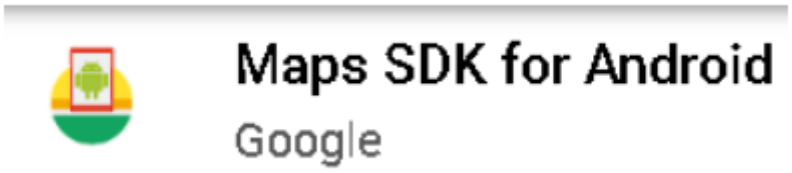


Figura 1.9. Logo Maps SDK [17]

Gracias a SDK de Maps (Figura 1.9.) para Android se puede visualizar mapas con la estructura de datos de Google Maps dentro de cualquier aplicación, automáticamente se accede a los servidores, se descarga datos, se visualiza y se tiene respuestas a los ítems del mapa, también se puede agregar marcadores y superponer todo a un mapa básico y crear una visualización propia del desarrollador. [17]

1.3.5.2. API de geo codificación

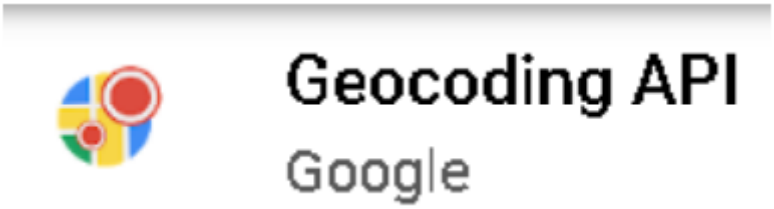


Figura 1.10. Logo de la API de geo codificación [18]

La geo codificación (Figura 1.10.) es convertir direcciones de lugares (como calles) en coordenadas geográficas de latitud y longitud lo que permitirá mostrar marcadores en varios lugares del mapa, también cuenta con geo codificación inversa que es el proceso de convertir latitud y longitud en una dirección legible común. [18]

1.3.5.3. API de geolocalización

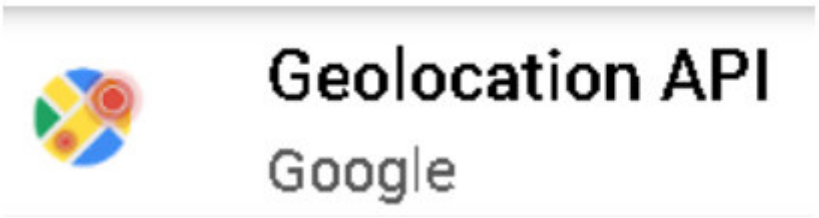


Figura 1.11. Logo de la API de geolocalización. [19]

Cuando se realiza una solicitud a esta API se obtiene como respuesta una ubicación con un radio de precisión muy bueno aproximadamente de 3 a 7 metros ya que se basa en información sobre torres de celdas y nodos Wifi. [19]

1.3.6. PLATAFORMA DE GOOGLE MAPS

Entre las varias herramientas de desarrollo esta plataforma es compatible con Android y iOS, consta de:

- Mapas dinámicos y estáticos que contienen vistas en 360°. Se puede personalizar lo mapas con diferentes imágenes, marcadores, líneas y colores.
- Rutas las cuales son líneas de un punto a otro y se lo puede implementar en tiempo real.
- Una base de datos de más de 100 millones de lugares, direcciones, números de teléfono y señales, también se puede utilizar tiempo real.

2. METODOLOGÍA

Este capítulo consta de cuatro partes: descripción de Android Studio IDE, creación del proyecto, creación de base de datos e implementación de todos los servicios. El IDE de Android Studio es un sistema de software que sirve para el diseño de aplicaciones y es necesario para iniciar con la programación del proyecto. En la creación del proyecto se indicará paso a paso a seguir para concluir el proyecto. En creación de base de datos se indicará a medida el funcionamiento básico y creación de Firebase Database. En implementación de los servicios se indicará paso a paso el incluir las diferentes APIs que se utilizarán en este proyecto.

2.1. REQUERIMIENTOS DE LA APLICACIÓN

Gracias a visitas, reuniones y en base a la información proporcionada por el grupo de Investigación Educativa (convocatoria INÉDITA) de la Escuela Politécnica Nacional se ha obtenido un listado de requerimientos funcionales y no funcionales que la aplicación cumplirá para su implementación. Tomando en cuenta que los requerimientos funcionales son las acciones que la aplicación tendrá y los requerimientos no funcionales será la seguridad, velocidad, utilidad, etc. de la aplicación. [20]

2.1.1. REQUERIMIENTOS FUNCIONALES

- Permitir la creación de usuarios.
- Permitir que el sistema guarde los datos de cada usuario.
- Permitir que el sistema autentique cada usuario dependiendo del rol.
- Permitir al usuario principal poder transcribir.
- Permitir al usuario principal poder guardar o descartar transcripciones.
- Permitir al usuario principal poder descargar un informe básico de utilidad.
- Permitir al usuario principal poder enviar y recibir mensajes con su usuario familiar.
- Permitir al usuario principal poder editar o eliminar transcripciones.
- Permitir al usuario principal poder ver su ubicación actual.
- Permitir al usuario principal poder enviar su ubicación actual por medio de WhatsApp.
- Permitir al usuario familiar poder ver los recorridos por fecha de su usuario principal.
- Permitir al usuario familiar poder enviar y recibir mensajes con su usuario principal.
- Permitir al usuario familiar poder descargar un informe básico de utilidad de su usuario principal.
- Permitir al usuario administrador poder ver información sobre la utilidad de la base de datos.
- Permitir al usuario administrador poder ver información simple de cada usuario.

- Permitir al usuario administrador poder crear usuarios administradores.

2.1.2. REQUERIMIENTOS NO FUNCIONALES

- Funcionar en dispositivos Android.
- La aplicación será desarrollada en Android Studio.
- Utilizar seguridad en Firebase.
- Utilizar al método de autenticación segura de Firebase.
- Presentar un tiempo de respuesta relativamente rápida (40ms – 200ms) entre el usuario y la aplicación.
- El prototipo deberá tener conexión a internet para su funcionamiento.

2.2. DISEÑO DE LA ARQUITECTURA DEL PROTOTIPO

En la figura 2.1 se indican las tres partes que ayuda al entendimiento de la arquitectura:

- Usuarios o Roles.
- Servicios.
- Base de Datos.

En la figura 2.1. se puede observar tres recuadros; un naranja ubicado a la izquierda, un verde en el centro y un azul en la derecha.

En el recuadro de color naranja se indica a los usuarios: administrador, principal y familiar los cuales utilizarán servicios de acuerdo a roles y permisos.

En la parte de color verde se indica los servicios, por ejemplo, inicio de sesión, eliminación de transcripción, etc.

En la parte de color azul se indica las tablas o nodos principales en la base de datos de donde el prototipo capturará y guardará información.

En la parte inferior izquierda de la figura 2.1. se indica las plataformas y tecnologías que se utiliza como son Java, PDF, Android, Firebase Database y Firebase Authentication.

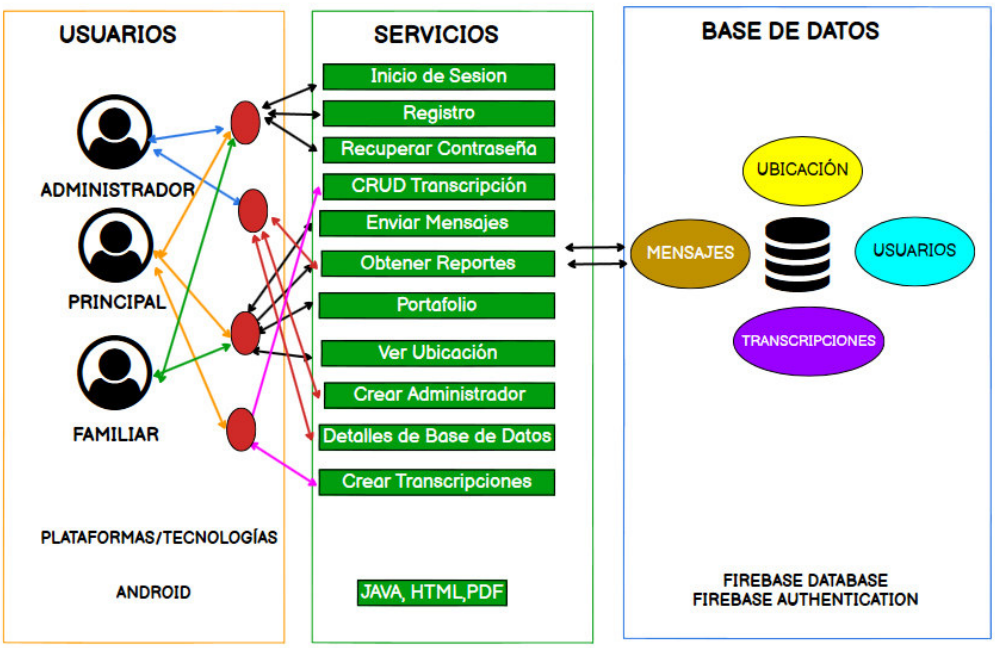


Figura 2.1. Usuarios, servicios y base de datos

2.3. HISTORIAS DE USUARIO

Son utilizadas para escribir las diferentes necesidades del cliente o usuario para el desarrollo de un producto software [21].

La gran diferencia entre *requerimientos* e *historias de usuario* es que los primeros son centralizados a ideas basadas en las operaciones del prototipo o sistema como tal, tendiendo a la especificación de que se debe permitir y que no; mientras que las segundas se centran en el resultado que se obtiene al realizar la acción detallada en cada una incluyendo una prioridad, es decir existe una relación entre las dos entidades que las hacen necesarias en este proyecto. Las historias de usuario están incluidas en el desarrollo ágil de software para tomar y detallar un poco los diferentes requisitos. [22]

2.3.1. FORMATO

En la Tabla 2.1. se puede visualizar el formato que tendrán las historias de usuario, a continuación, se indica un mayor detalle de los campos que tendrán las historias de usuario:

1. **Identificación (ID):** Este campo indicará el número de secuencia para identificar cada HU (Historia Usuario), por lo general lleva las siglas HU seguida de un numero (3 o 4 cifras), por ejemplo: HU011.
2. **Prioridad:** Este campo indicará la importancia que tiene dicha HU referente a todas.
3. **Nombre de historia:** Este campo es un nombre para identificar en ideas al requerimiento que el usuario o cliente solicitó.

4. **Descripción:** Este campo contiene un pequeño detalle del requerimiento.

Tabla 2.1. Formato de historia de usuario

HISTORIA DE USUARIO	
ID:	**Prioridad:**
Nombre de historia:	
Descripción:	

Para el campo prioridad es necesario conocer los niveles de prioridad que pueden ser asignados a cada historia de usuario, en todos los casos esta prioridad es definida por el cliente, pero para el desarrollo de este prototipo se ha tomado en cuenta la dificultad y el tiempo de cada historia de usuario y se ha asignado la prioridad por el desarrollador de la aplicación.

- **Nivel muy alto o urgente:** nivel 5.
- **Nivel alto o importante:** nivel 4.
- **Nivel medio o moderado:** nivel 3.
- **Nivel bajo o tolerable:** nivel 2.
- **Nivel muy bajo o trivial:** nivel 1.

2.3.2. HISTORIAS DE USUARIO

A continuación, se detallan la historia de usuario HU001 para mayor entendimiento y a continuación el total de historias de usuario.

La tabla 2.2. indica la creación de credenciales para poder utilizar todas las características que contiene Firebase, sin este paso no se puede activar la cuenta Firebase lo que impide la creación de la base de datos en tiempo real.

Tabla 2.2. Historia de usuario HU001

HISTORIA DE USUARIO	
ID: HU001	**Prioridad:** 3

Nombre de historia: Crear credenciales para iniciar Firebase.
Descripción: Se creará las credenciales necesarias con los requerimientos que Google solicite.

Tabla 2.3. Historia de usuario HU002

HISTORIA DE USUARIO	
ID: HU002	**Prioridad:** 3
Nombre de historia: Crear credenciales para Google Cloud Plataform	
Descripción: Se creará las credenciales necesarias con los requerimientos que Google solicite.	

Tabla 2.4. Historia de usuario HU003

HISTORIA DE USUARIO	
ID: HU003	**Prioridad:** 2
Nombre de historia: Configurar base de datos	
Descripción: Se configura los permisos necesarios para el correcto funcionamiento de la base de datos en función del prototipo.	

Tabla 2.5. Historia de usuario HU004

HISTORIA DE USUARIO	
ID: HU004	**Prioridad:** 5
Nombre de historia: Crear base de datos	

Descripción: Se crea la base de datos con los nodos necesarios para el correcto funcionamiento de la base de datos en función del prototipo.

Tabla 2.6. Historia de usuario HU005

HISTORIA DE USUARIO	
ID: HU005	**Prioridad:** 5
Nombre de historia: Sincronizar APIs con proyecto	
Descripción: Sincronizar todo el proyecto con las diferentes APIs que se utilizarán.	

Tabla 2.7. Historia de usuario HU006

HISTORIA DE USUARIO	
ID: HU006	**Prioridad:** 2
Nombre de historia: Generar respaldo Git para actualizaciones	
Descripción: Crear un respaldo utilizando Git Hub y Git para manejo de versiones.	

Tabla 2.8. Historia de usuario HU007

HISTORIA DE USUARIO	
ID: HU007	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Registro	
Descripción: Crear interfaz para dar toda la funcionalidad de registro o creación de usuario.	

Tabla 2.9. Historia de usuario HU008

HISTORIA DE USUARIO	
ID: HU008	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Inicio de sesión	
Descripción: Crear interfaz para dar toda la funcionalidad de inicio de sesión a los diferentes roles.	

Tabla 2.10. Historia de usuario HU009

HISTORIA DE USUARIO	
ID: HU009	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Recuperar contraseña	
Descripción: Crear interfaz para dar toda la funcionalidad para poder recuperar o cambiar la contraseña de inicio de sesión.	

Tabla 2.11. Historia de usuario HU010

HISTORIA DE USUARIO	
ID: HU010	**Prioridad:** 5
Nombre de historia: Crear interfaz para Menú Desplegable Principal	
Descripción: Generar interfaz con todos los atributos necesarios para el rol Principal de nuestro prototipo.	

Tabla 2.12. Historia de usuario HU011

HISTORIA DE USUARIO

ID: HU011	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Sincronización Familiar	
Descripción: Crear interfaz donde el usuario del rol Principal podrá sincronizarse con un usuario del rol Familiar.	

Tabla 2.13. Historia de usuario HU012

HISTORIA DE USUARIO	
ID: HU012	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Eliminar Sincronización	
Descripción: Crear interfaz donde el usuario del rol Principal podrá quitar su sincronización con un usuario del rol Familiar.	

Tabla 2.14. Historia de usuario HU013

HISTORIA DE USUARIO	
ID: HU013	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Transcripción	
Descripción: Crear interfaz donde el usuario del rol Principal podrá iniciar, pausar, continuar, descartar y guardar su transcripción.	

Tabla 2.15. Historia de usuario HU014

HISTORIA DE USUARIO	
ID: HU014	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Portafolio	

Descripción: Crear interfaz donde el usuario del rol Principal podrá ver una lista de sus transcripciones guardadas previamente.

Tabla 2.16. Historia de usuario HU015

HISTORIA DE USUARIO	
ID: HU015	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Edición de Transcripción	
Descripción: Crear interfaz donde el usuario del rol Principal podrá ingresar a una transcripción, podrá editarla, descartar cambios o eliminarla.	

Tabla 2.17. Historia de usuario HU016

HISTORIA DE USUARIO	
ID: HU016	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Ubicación	
Descripción: Crear interfaz donde el usuario del rol Principal podrá ver su ubicación y mediante un botón podrá enviarla a un contacto de su aplicación WhatsApp.	

Tabla 2.18. Historia de usuario HU017

HISTORIA DE USUARIO	
ID: HU017	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Reporte	

Descripción: Crear interfaz donde el usuario del rol Principal podrá ver un reporte y guardarlo en su celular en formato PDF.

Tabla 2.19. Historia de usuario HU018

HISTORIA DE USUARIO	
ID: HU018	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Chat	
Descripción: Crear interfaz donde el usuario del rol Principal podrá enviar y recibir mensajes con su usuario Familiar sincronizado.	

Tabla 2.20. Historia de usuario HU019

HISTORIA DE USUARIO	
ID: HU019	**Prioridad:** 4
Nombre de historia: Crear interfaz para Menú Desplegable Familiar	
Descripción: Generar interfaz con todos los atributos necesarios para el rol Familiar de nuestro prototipo.	

Tabla 2.21. Historia de usuario HU020

HISTORIA DE USUARIO	
ID: HU020	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Ver Ubicación	
Descripción: Crear interfaz donde el usuario del rol Familiar podrá ver el recorrido por fechas de los puntos donde estuvo su usuario Principal sincronizado.	

Tabla 2.22. Historia de usuario HU021

HISTORIA DE USUARIO	
ID: HU021	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Reporte (Familiar)	
Descripción: Crear interfaz donde el usuario del rol Familiar podrá ver un reporte de su usuario Principal sincronizado y guardarlo en su celular en formato PDF.	

Tabla 2.23. Historia de usuario HU022

HISTORIA DE USUARIO	
ID: HU022	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Chat (Familiar)	
Descripción: Crear interfaz donde el usuario del rol Familiar podrá enviar y recibir mensajes con su usuario Principal sincronizado.	

Tabla 2.24. Historia de usuario HU023

HISTORIA DE USUARIO	
ID: HU023	**Prioridad:** 4
Nombre de historia: Crear interfaz para Menú Desplegable Administrador	
Descripción: Generar interfaz con todos los atributos necesarios para el rol Administrador de nuestro prototipo.	

Tabla 2.25. Historia de usuario HU024

HISTORIA DE USUARIO	
ID: HU024	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Detalles	
Descripción: Generar interfaz con todos los atributos necesarios para visualizar una lista de los usuarios Principales de la base de datos con su información básica.	

Tabla 2.26. Historia de usuario HU025

HISTORIA DE USUARIO	
ID: HU025	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Detalles de Usuario en rol Administrador	
Descripción: Generar interfaz con todos los atributos necesarios para visualizar información básica de cada usuario Principal alojado en la base de datos previo a su selección.	

Tabla 2.27. Historia de usuario HU026

HISTORIA DE USUARIO	
ID: HU026	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Reporte General	
Descripción: Generar interfaz con todos los atributos necesarios para visualizar información específica de la base de datos.	

Tabla 2.28. Historia de usuario HU027

HISTORIA DE USUARIO	
ID: HU027	**Prioridad:** 4
Nombre de historia: Crear interfaz para modulo Crear Administrador	
Descripción: Generar interfaz con todos los atributos necesarios para crear un nuevo usuario Administrador.	

Tabla 2.29. Historia de usuario HU028

HISTORIA DE USUARIO	
ID: HU028	**Prioridad:** 3
Nombre de historia: Codificar botones con características propias	
Descripción: Codificar botones que se utilizarán varias veces para ahorro de tiempo de programación.	

Tabla 2.30. Historia de usuario HU029

HISTORIA DE USUARIO	
ID: HU029	**Prioridad:** 3
Nombre de historia: Codificar frases para textos repetitivos	
Descripción: Codificar frases que se utilizarán varias veces para ahorro de tiempo de programación.	

Tabla 2.31. Historia de usuario HU030

HISTORIA DE USUARIO

ID: HU030	**Prioridad:** 5
Nombre de historia: Codificar Menús Desplegables	
Descripción: Codificar los menús necesarios para cada rol.	

Tabla 2.32. Historia de usuario HU031

HISTORIA DE USUARIO	
ID: HU031	**Prioridad:** 5
Nombre de historia: Codificar Rol Principal Completo	
Descripción: Codificar el rol Principal con todas sus características y ejecutarlas.	

Tabla 2.33. Historia de usuario HU032

HISTORIA DE USUARIO	
ID: HU032	**Prioridad:** 5
Nombre de historia: Codificar Rol Familiar Completo	
Descripción: Codificar el rol Familiar con todas sus características y ejecutarlas.	

Tabla 2.34. Historia de usuario HU033

HISTORIA DE USUARIO	
ID: HU033	**Prioridad:** 5
Nombre de historia: Codificar Rol Administrador Completo	
Descripción: Codificar el rol Administrador con todas sus características y ejecutarlas.	

Tabla 2.35. Historia de usuario HU034

HISTORIA DE USUARIO	
ID: HU034	**Prioridad:** 4
Nombre de historia: Codificar Alertas	
Descripción: Codificar todas las alertas necesarias para confirmar acciones.	

Tabla 2.36. Historia de usuario HU035

HISTORIA DE USUARIO	
ID: HU035	**Prioridad:** 5
Nombre de historia: Codificar PDF en roles necesarios	
Descripción: Codificar todos los campos necesarios para poder crear y descargar un archivo PDF el cual contenga información de transcripciones guardadas en la base de datos.	

Tabla 2.37. Historia de usuario HU036

HISTORIA DE USUARIO	
ID: HU036	**Prioridad:** 5
Nombre de historia: Controlar pedidos a base de datos	
Descripción: Controlar repeticiones de pedidos a la base de datos ya que esta trabaja en tiempo real.	

Tabla 2.38. Historia de usuario HU037

HISTORIA DE USUARIO

ID: HU037	**Prioridad:** 5
Nombre de historia: Controlar cache en prototipo instalado	
Descripción: Controlar cache de aplicación ya que la base de datos trabaja en tiempo real.	

2.3.3. PRODUCT BACKLOG LIST

En la Tabla 2.4. se puede observar la Product backlog list con su ID, el nombre o detalle y el *Sprint* al que pertenece, estos requerimientos se relacionan con las historias de usuario indicadas en la parte superior.

Para recordar cada *Sprint* debe durar entre 1 a 4 semanas, para el prototipo cada *Sprint* tardará de 7 a 10 días para finalizarlo con las pruebas pertinentes.

Tabla 2.39. Product backlog list

ID	Nombre	Sprint
PBL001	Crear credenciales para iniciar Firebase	1
PBL002	Crear credenciales para Google Cloud Plataform	1
PBL003	Configurar base de datos	2
PBL004	Crear base de datos	2
PBL005	Sincronizar APIs con proyecto	2
PBL006	Generar respaldo Git para actualizaciones	2
PBL007	Crear interfaz para modulo Registro	3
PBL008	Crear interfaz para modulo Inicio de sesión	3
PBL009	Crear interfaz para modulo Recuperar contraseña	3
PBL010	Crear interfaz para Menú Desplegable Principal	4
PBL011	Crear interfaz para modulo Sincronización Familiar	4

PBL012	Crear interfaz para modulo Eliminar Sincronización	4
PBL013	Crear interfaz para modulo Transcripción	4
PBL014	Crear interfaz para modulo Portafolio	4
PBL015	Crear interfaz para modulo Edición de Transcripción	4
PBL016	Crear interfaz para modulo Ubicación	4
PBL017	Crear interfaz para modulo Reporte	4
PBL018	Crear interfaz para modulo Chat	4
PBL019	Crear interfaz para Menú Desplegable Familiar	5
PBL020	Crear interfaz para modulo Ver Ubicación	5
PBL021	Crear interfaz para modulo Reporte (Familiar)	5
PBL022	Crear interfaz para modulo Chat (Familiar)	5
PBL023	Crear interfaz para Menú Desplegable Administrador	6
PBL024	Crear interfaz para modulo Detalles	6
PBL025	Crear interfaz para modulo Detalles de Usuario en rol Administrador	6
PBL026	Crear interfaz para modulo Reporte General	6
PBL027	Crear interfaz para modulo Crear Administrador	6
PBL028	Codificar botones con características propias	7
PBL029	Codificar frases para textos repetitivos	7
PBL030	Codificar Menús Desplegables	8
PBL031	Codificar Rol Principal Completo	9
PBL032	Codificar Rol Familiar Completo	10

PBL033	Codificar Rol Administrador Completo	11
PBL034	Codificar Alertas	12
PBL035	Codificar PDF en roles necesarios	12
PBL036	Controlar pedidos a base de datos	13
PBL037	Controlar cache en prototipo instalado	13

2.3.4. CASOS DE USO

Gracias a los requerimientos para el prototipo se determinó tres tipos de roles para usuarios: Administrador, Principal y Familiar.

Las especificaciones de cada sección de la Figura 2.2, Figura 2.3 y Figura 2.4 se detallan a continuación:

Ingresar menú Administrador: Permite ingresar al menú desplegable del rol Administrador.

Validación de identidad: Permite comprobar datos de autenticación en la base de datos.

Creación de usuario Administrador: Permite crear un nuevo usuario perteneciente al rol Administrador y guardar sus credenciales en la base de datos.

Consultar detalles: Permite obtener detalles desde la base de datos.

Consultar lista de usuarios Principal: Permite hacer un llamado a la base de datos para consultar todos los usuarios Principal.

Consultar detalle de usuario: Permite hacer llamado a diferentes puntos la base de datos para consultar detalles propios del usuario seleccionado.

Consultar reporte general: Permite obtener varios detalles básicos de la base de datos.

Consultar lista de usuarios Administrador: Permite hacer un llamado a la base de datos para consultar todos los usuarios Administrador.

Consultar lista de usuarios Familiar: Permite hacer un llamado a la base de datos para consultar todos los usuarios Familiar.

Consultar lista de transcripciones: Permite hacer un llamado a la base de datos para consultar todas las transcripciones de todos los usuarios.

Consultar lista de mensajes: Permite hacer un llamado a la base de datos para consultar todos los mensajes de todos los usuarios.

Ingresar menú Familiar: Permite ingresar al menú desplegable del rol Familiar.

Consultar reporte: Permite obtener varios detalles básicos de la base de datos referente al usuario registrado.

Guardar reporte: Permite capturar datos pertenecientes al usuario y realizar una conexión entre la aplicación y el dispositivo para guardar aquellos datos.

Crear PDF: Permite crear un archivo PDF con estructura programada dentro de la aplicación.

Chatear: Permite comunicarse con otro usuario.

Consultar mensajes propios: Permite hacer un llamado a la base de datos y capturar todos los mensajes sean estos recibidos o enviados.

Recibir mensaje: Permite crear la acción de recibir mensajes instantáneamente.

Enviar mensaje: Permite crear la acción de enviar mensajes instantáneamente.

Consultar ubicación: Permite llamar al mapa para dibujar puntos.

Consultar lista de ubicaciones: Permite hacer un llamado a la base de datos y capturar los puntos de ubicación de un usuario en particular.

Ingresar menú Principal: Permite ingresar al menú desplegable del rol Principal.

Sincronizar con familiar: Permite crear un enlace entre un usuario Principal y un Familiar.

Seleccionar Familiar: Permite seleccionar un elemento de la lista de usuarios Familiar y cambiar estados de atributos.

Eliminar sincronización: Permite eliminar un enlace entre un usuario Principal y un Familiar.

Consultar usuario Familiar: Permite hacer llamado a diferentes puntos la base de datos para consultar detalles propios del usuario Familiar que tiene sincronismo.

Transcribir: Permite visualizar opciones que el usuario puede utilizar para realizar un cambio de voz a texto entre estos:

- Iniciar transcripción.
- Guardar transcripción, siempre se debe tener un título.
- Pausar transcripción.
- Continuar transcripción.
- Aumentar tamaño de letra.
- Minimizar tamaño de letra.

En la Figura 2.2. se detalla el rol Administrador quien es el encargado de ver detalles generales de cada usuario, además podrá ver el informe de utilización de cada usuario y un informe conjunto de cómo se está usando la aplicación. [23]

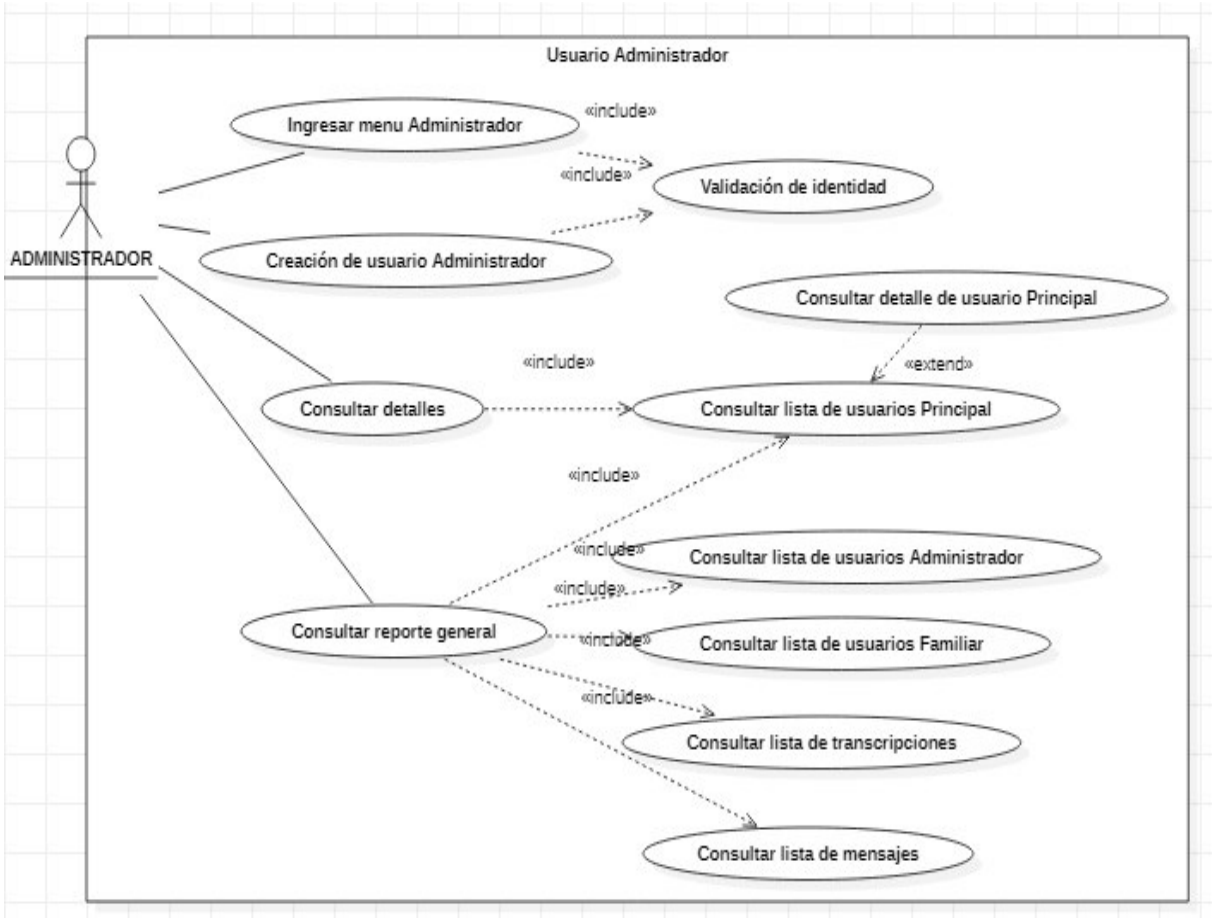


Figura 2.2. Casos de uso del rol Administrador

Como se observa en la Figura 2.3. se encuentra el rol Familiar quien es el encargado de visualizar a detalle la ubicación y el recorrido del usuario Principal si este lo permite, podrá ver un informe de como utiliza la aplicación el usuario Principal al que esté asignado y podrá enviar y recibir mensajes.

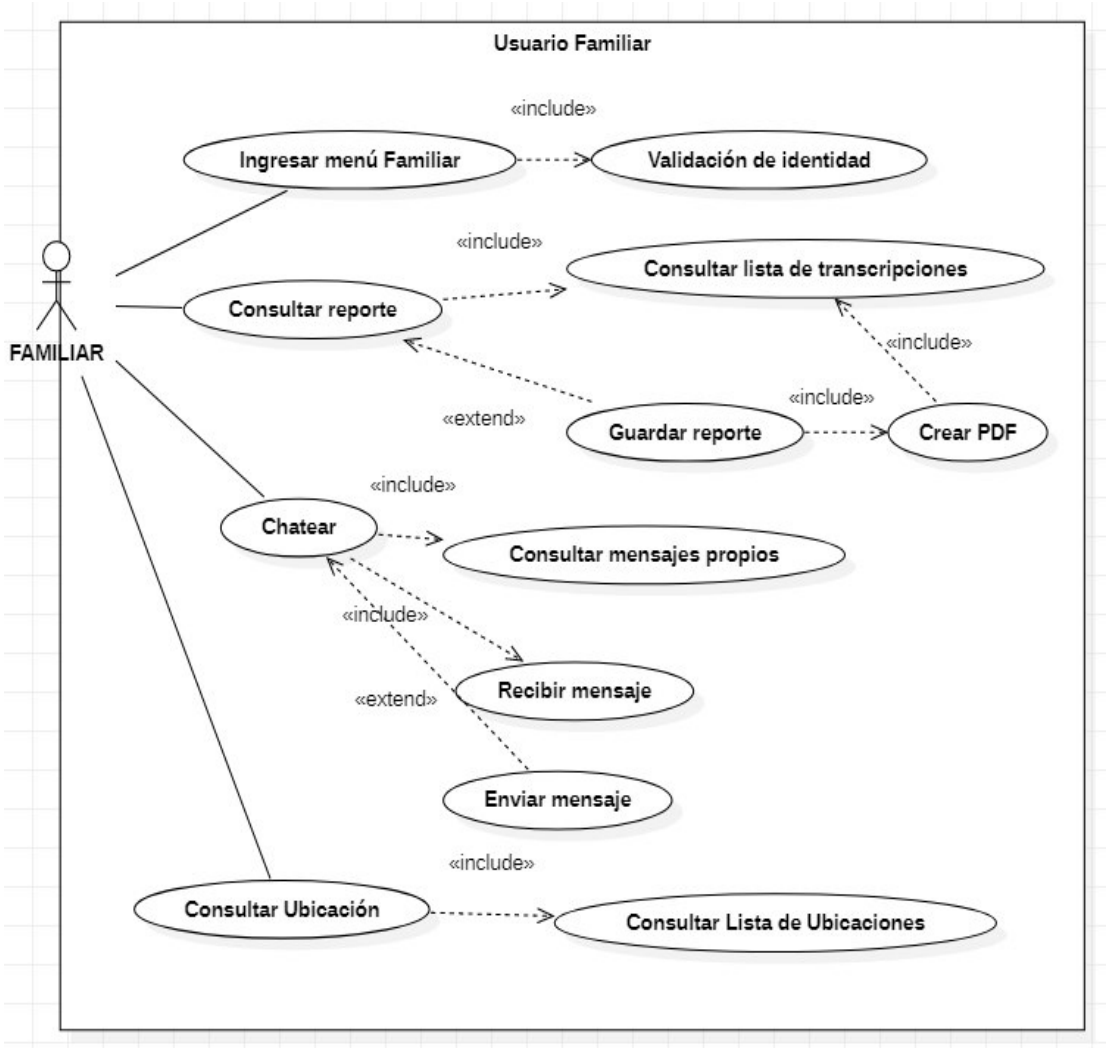


Figura 2.3. Casos de uso del rol Familiar

Como se indica en la Figura 2.4. se detalla el rol Principal quien inicia la transcripción y podrá guardar cada una según el considere, o eliminar las previamente guardadas, podrá mostrar la ubicación actual a su familiar asignado y además enviar la ubicación a un contacto dentro de su aplicación WhatsApp, dependiendo como utilice la aplicación podrá ver un informe o un reporte propio, si desea comunicarse con su familiar a través de texto tendrá una opción de envío y recepción de mensajes.

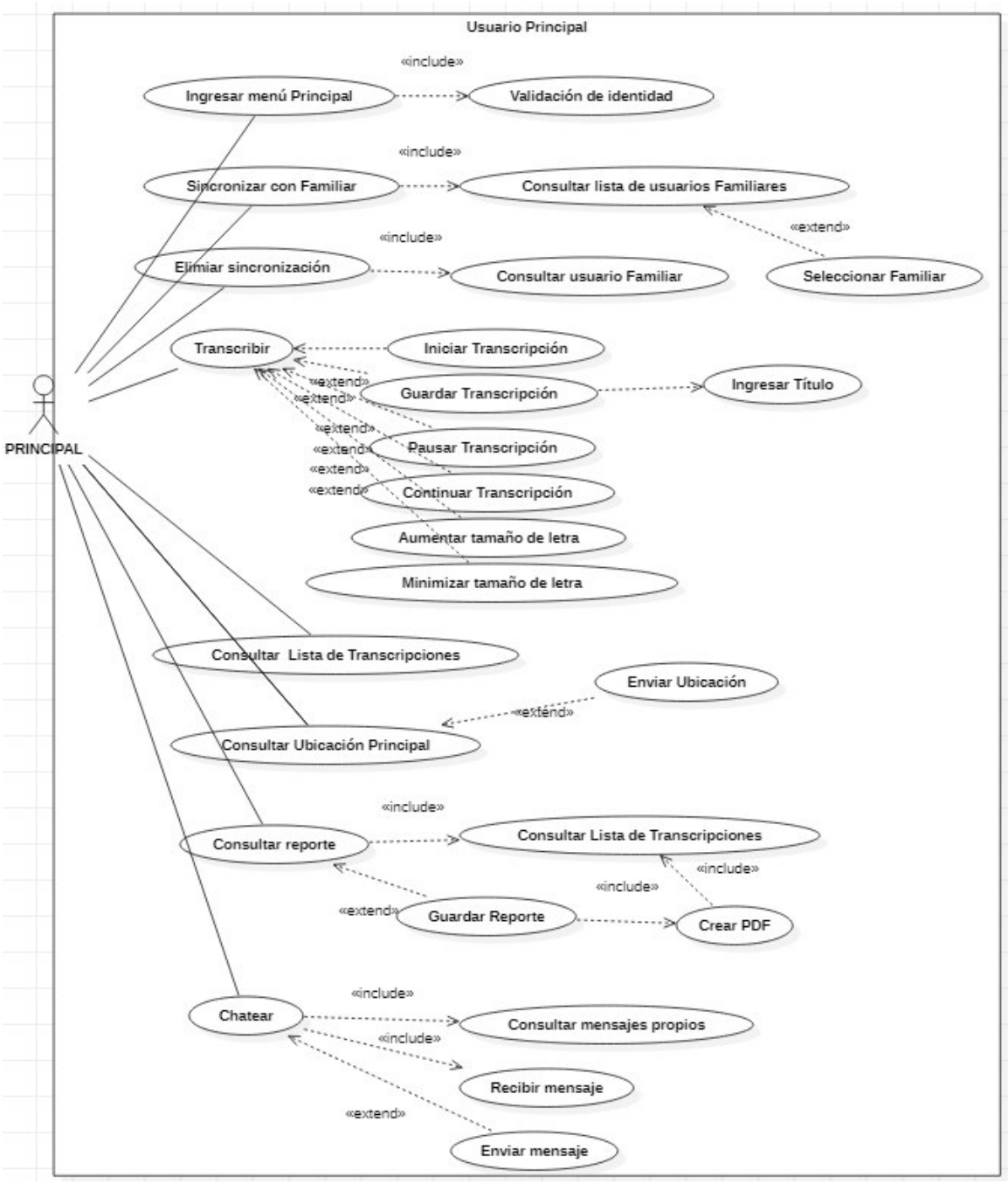


Figura 2.4. Casos de uso del rol Principal

2.3.5. DIAGRAMA DE CLASES

En la Figura 2.5. se muestra el diagrama de clases donde se indica la relación estática del proyecto a diseñar, es decir, contiene atributos y operaciones que permiten entender el funcionamiento base del proyecto. [23]

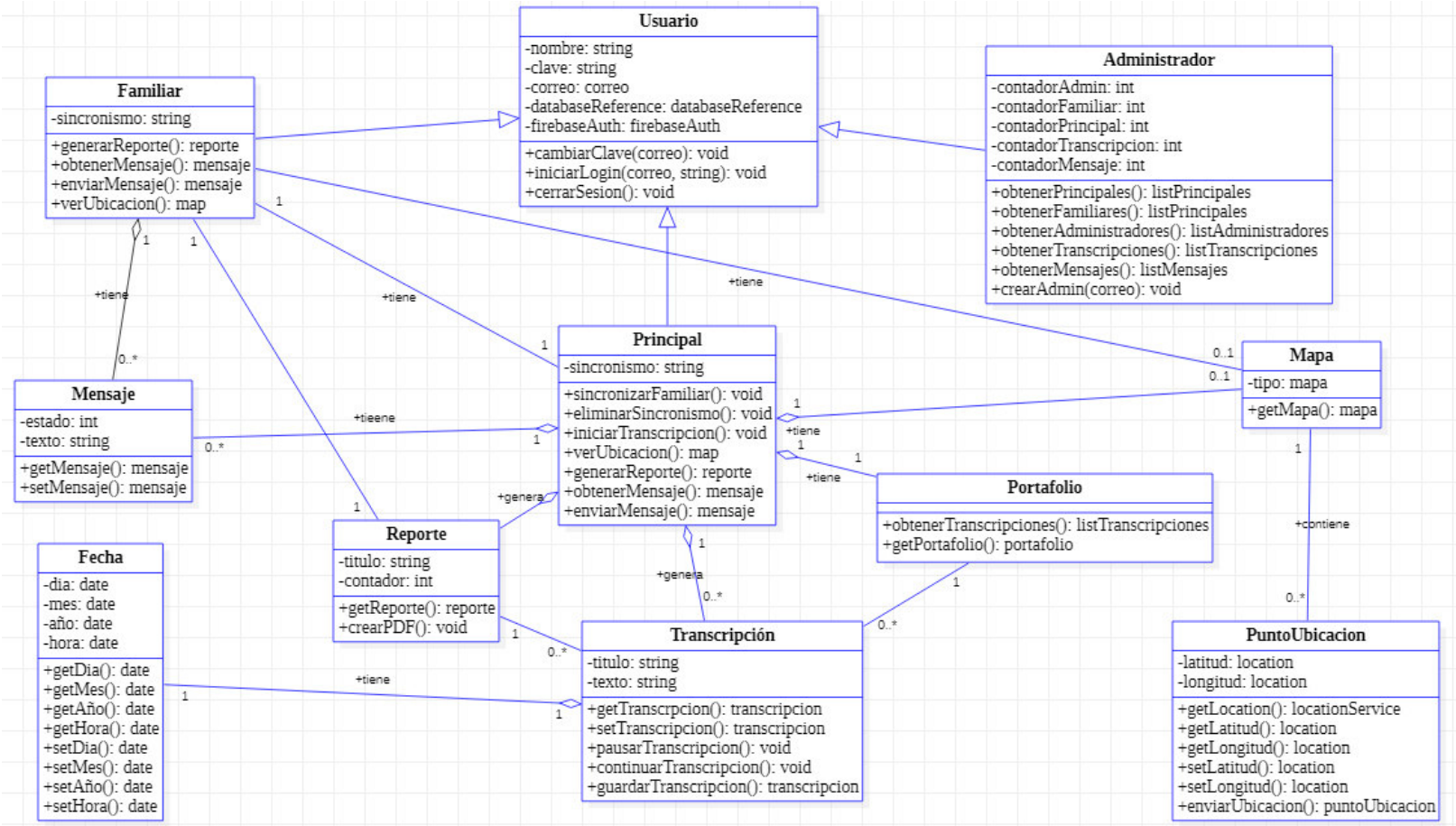


Figura 2.5. Diagrama de Clases [24]

A continuación, se detalla las partes individuales del diagrama de clases mostrado en la figura 2.5:

- **Usuario:** Esta clase permite gestionar atributos y operaciones que pueden realizar todos los usuarios.
- **Familiar:** Esta clase permite gestionar atributos y operaciones que puede realizar el usuario Familiar como enviar mensajes o ver ubicación de su usuario sincronizado.
- **Administrador:** Esta clase permite gestionar atributos y operaciones que puede realizar el usuario Administrador como un reporte general de la base de datos.
- **Principal:** Esta clase permite gestionar atributos y operaciones que puede realizar el usuario Principal como transcribir, enviar mensajes, enviar ubicación por WhatsApp, ver ubicación actual, ver reporte, informarse, eliminar, guardar y editar sus transcripciones, sincronizar y eliminar sincronismo.
- **Mapa:** Esta clase permite gestionar el formato del mapa que se desea y cuantos puntos o marcadores se quiere mostrar.
- **PuntoUbicacion:** Esta clase permite capturar una locación por medio de longitud y latitud para enviarla a la clase Mapa.
- **Reporte:** Esta clase permite obtener un reporte de transcripciones sea para el usuario Principal o Familiar, se podrá crear el PDF.
- **Transcripción:** Esta clase permite gestionar todo lo referente a poder transcribir de voz a texto, ya sea iniciar, pausar, continuar, guardar y aumentar o disminuir el texto del resultado de la transcripción.
- **Fecha:** Esta clase tendrá mucha importancia para el momento de guardar la transcripción en la base de datos ya que contiene un día, mes, año y hora que permitirá tener un orden y evitar posibles errores.
- **Portafolio:** Esta clase permite obtener una lista de todas las transcripciones pertenecientes al usuario y también brinda la opción de gestionar cada transcripción en el usuario Principal.
- **Mensaje:** Esta clase permite gestionar todo lo referente a envío y recepción de mensajes entre dos usuarios previamente sincronizados.

2.3.6. DIAGRAMA DE NODOS

El diagrama de nodos ayudará al diseño de la base de datos, cada nodo debe tener una relación no dependiente por más de dos sub nodos, es decir para el funcionamiento óptimo del llamado desde la aplicación a la base de datos y viceversa es necesario que la mayor cantidad de nodos sean independientes.

Como se indica en la Figura 2.6. se tiene 4 nodos principales que tendrán dependencia del nodo “USUARIOS”, es decir que existirán siempre y cuando el usuario tome acciones dentro de la aplicación.

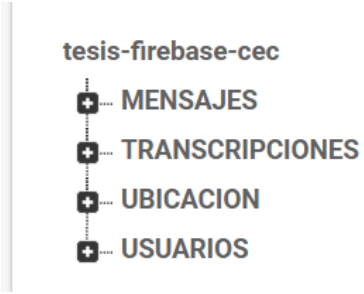


Figura 2.6. Nodos principales

Como se puede observar en la Figura 2.7. se organiza el nodo principal “MENSAJES”, con nodo a nivel dos “id usuario”, nodo a nivel tres “id mensaje” y nodos nivel cuatro “estado” y “texto”; no se aprecia el nombre de los atributos del id ya que *Firebase* puede guardar información en forma consecutiva y crea un id único.

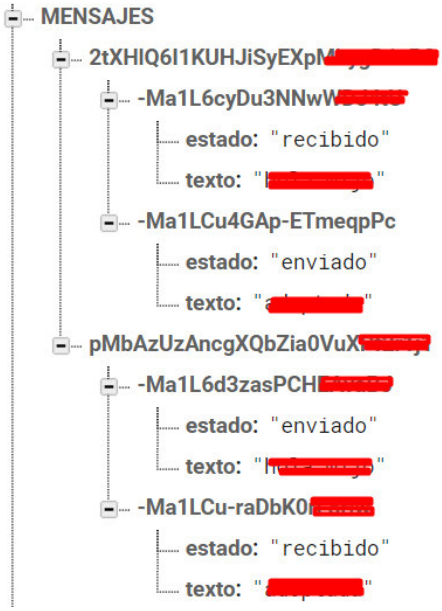


Figura 2.7. Nodo mensajes

En la Figura 2.8. se organiza el nodo principal “TRANSCRIPCIONES”, con nodo a nivel dos “id usuario”, nodo a nivel tres “id título” y nodos nivel cuatro “fecha”, “titulo” y “transcripción”.

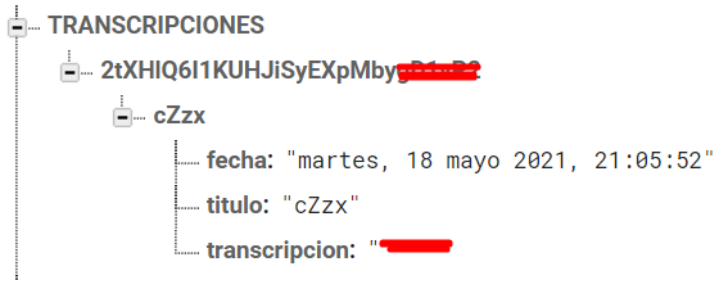


Figura 2.8. Nodo transcripciones

Como se indica en la Figura 2.9. se organiza el nodo principal “UBICACION”, con nodo a nivel dos “id usuario”, nodo a nivel tres “id punto” y nodos nivel cuatro “fecha”, “latitud” y “longitud”.

Figura 2.9. Nodo ubicación

Como se puede observar en la Figura 2.10. se organiza el nodo principal “USUARIOS” y con nodos a nivel dos “ADMINISTRADOR”, “FAMILIAR” y “PRINCIPAL”.

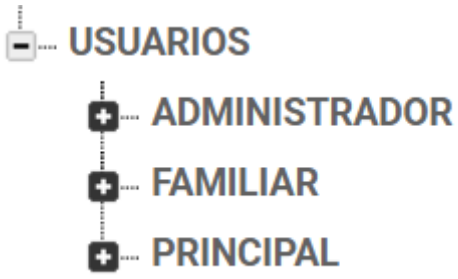


Figura 2.10. Nodo usuarios

En la Figura 2.11. se organiza el sub nodo principal “ADMINISTRADOR”, con nodo a nivel dos “id usuario”, y nodos nivel tres “apodo”, “clave”, “correo”, “sincronismo” (cuando exista sincronismo con otro usuario se tomará el id” y “id propio”.

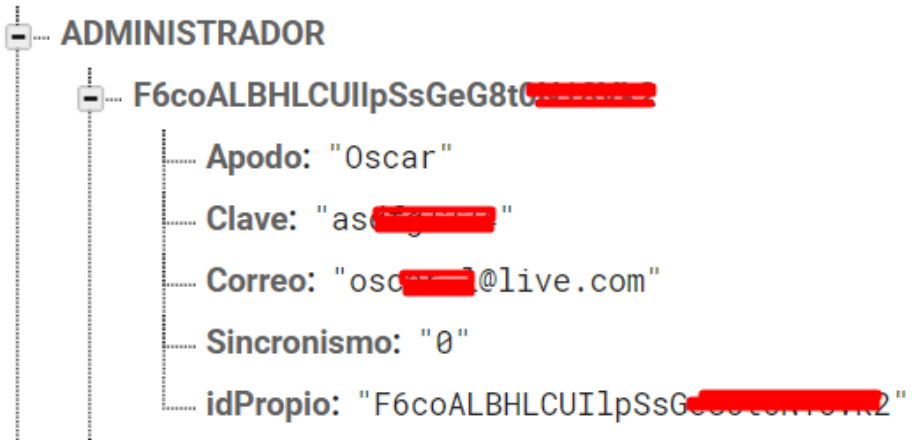


Figura 2.11. Sub nodo usuario

2.4. DISEÑO DE INTERFACES DE LA APLICACIÓN ANDROID

Para el diseño de sketches (escenas) se utilizó la aplicación de escritorio para Windows Balsamiq[4], estas interfaces de usuario del prototipo están conformadas por varias escenas generales e independientes para cada rol:

Generales

- Inicio de Sesión.
- Registrarse.
- Recuperar Contraseña.

Rol Administrador

- Menú Desplegable Administrador.
- Detalles.
- Detalles de Usuario en rol Administrador.
- Reporte General.
- Crear Administrador.

Rol Principal

- Menú Desplegable Principal.
- Sincronización Familiar.
- Eliminar Sincronización.
- Transcripción.

[4] **Balsamiq:** Programa que sirve para crear *sketches* (dibujos o bocetos) que permiten visualizar de manera fácil como quedarán las interfaces en una aplicación.

- Portafolio.
- Edición de Transcripción.
- Ubicación.
- Reporte.
- Chat.

Rol Familiar

- Menú Desplegable Familiar.
- Ver Ubicación.
- Reporte.
- Chat.

A continuación, se indica las escenas *Generales*

2.4.1. ESCENA "INICIO DE SESIÓN"

Como se muestra en la Figura 2.12. esta es la escena inicial cuando se instala la aplicación. Esta escena está conformada por:

- Edit Text Email: Este elemento permitirá ingresar el correo electrónico independientemente del rol de usuario.
- Edit Text Clave: Este elemento permitirá ingresar la clave correspondiente al correo electrónico.
- Botón Ingresar: Permite abrir la escena del menú para cada usuario.
- Botón Registrarse: Permite abrir la escena para registrar un usuario nuevo.
- Botón Recuperar Contraseña: Permite abrir la escena para recuperar o cambiar la clave.

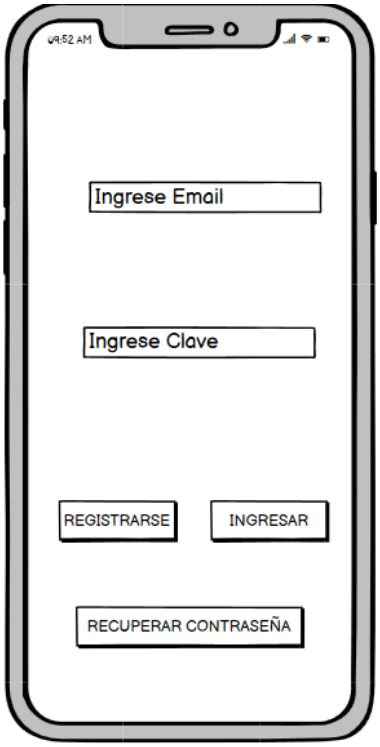


Figura 2.12. Escena Inicio de Sesión

2.4.2. ESCENA "REGISTRARSE"

Como se observa en la Figura 2.13. esta escena se abre al seleccionar al botón *Registrarse* en el Inicio de Sesión, en la cual se puede crear un nuevo usuario. Esta escena está conformada por:

- Botón Radio: Permite seleccionar el usuario a crear.
- Text Apodo o Nombre: Permite ingresar el apodo o nombre del usuario a crear.
- Text Email: Permite ingresar el correo.
- Text Clave: Permite ingresar la contraseña.
- Text Repetir Clave: Permite comprobar la contraseña ingresada.
- Botón Continuar: Permite registrar el usuario creado.
- Diálogo de Alerta: Permite confirmar el correo del usuario a crear.

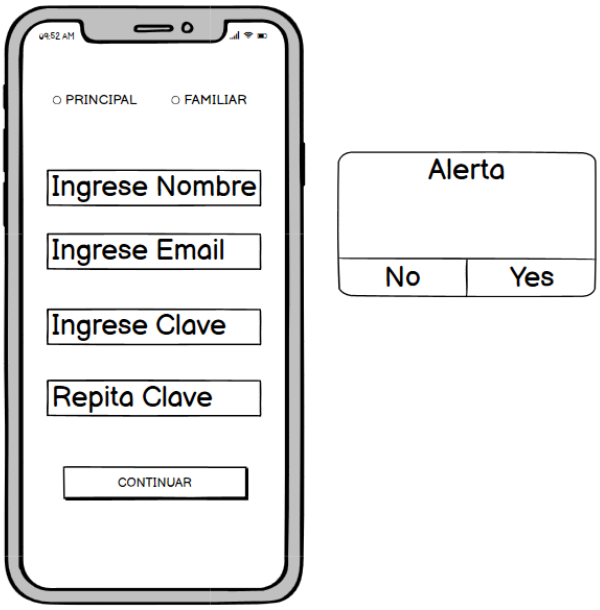


Figura 2.13. Escena Registrarse

2.4.3. ESCENA "RECUPERAR CONTRASEÑA"

Como podemos ver en la Figura 2.14. esta escena se abre al seleccionar al botón *Recuperar* Contraseña en el Inicio de Sesión, en la cual se puede cambiar o recuperar contraseña. Esta escena está conformada por:

- Text Ingrese correo: Permite comprobar el correo ingresado.
- Botón Confirmar: Permite enviar un mail al correo indicado en Text Ingrese correo.

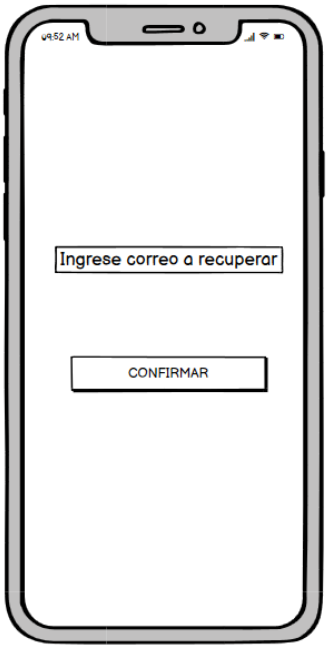


Figura 2.14. Escena Recuperar Contraseña

A continuación, se indica las escenas para el *Rol Administrador*

2.4.4. ESCENA "MENÚ DESPLEGABLE ADMINISTRADOR"

Se puede ver en la Figura 2.15. que esta escena se abre al seleccionar al botón *Ingresar* en el Inicio de Sesión y un correo perteneciente al rol Administrador, en la cual se obtiene todas las acciones posibles para el rol Administrador. Esta escena está conformada por:

- Imagen: Escudo de la Escuela Politécnica Nacional.
- Text Nombre: Nombre o apodo del usuario.
- Text Correo: Correo del usuario.
- Text Rol: Texto del rol al que pertenece el usuario.
- Text Detalles: Texto que al seleccionar enviará a escena Detalles.
- Text Reporte General: Texto que al seleccionar enviará a escena Reporte General.
- Text Crear Administrador: Texto que al seleccionar enviará a escena Crear Administrador.
- Text Cerrar Sesión: Texto que al seleccionar cerrará sesión y enviará a escena Inicio de Sesión.

Figura 2.15. Escena Menú Desplegable Administrador

2.4.5. ESCENA “DETALLES”

Como se indica en la Figura 2.16. esta escena se abre al seleccionar el Text *Detalles* en el Menú Desplegable Administrador, en la cual se obtiene una lista de todos los usuarios Principales. Esta escena está conformada por:

- Text Información: Informa que se debe seleccionar un usuario.
- Lista: Lista de usuarios Principales.

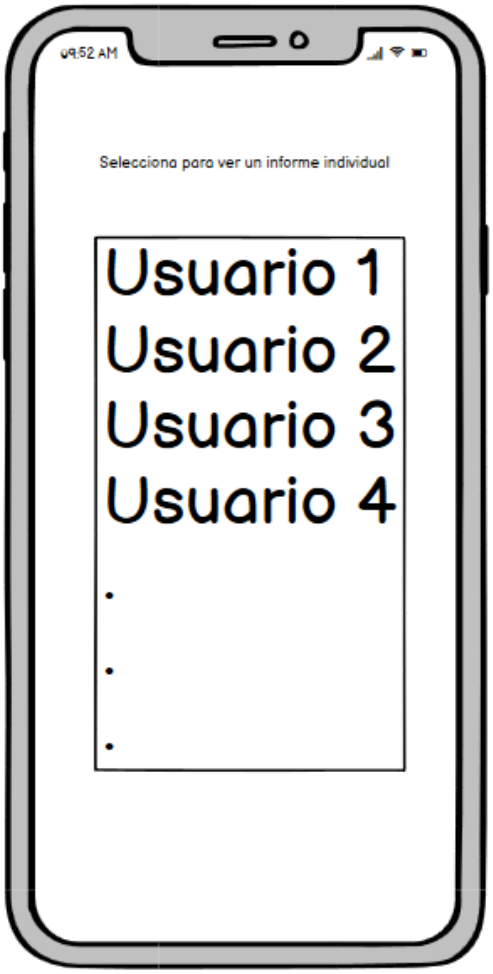


Figura 2.16. Escena Detalles

2.4.5.1. Escena “Detalles de Usuario en rol Administrador”

Se puede ver en la Figura 2.17. que esta escena se abre al seleccionar el usuario de la Lista en la escena *Detalles*, en la cual se obtiene información del usuario seleccionado. Esta escena está conformada por:

- Text usuario seleccionado: Texto usuario seleccionado.
- Text nombre usuario seleccionado: Texto donde se indica el nombre del usuario seleccionado.

- Text correo: Texto correo.
- Text correo usuario seleccionado: Texto donde se indica el correo del usuario seleccionado.
- Text Sincronismo: Información de sincronismo.
- Text Transcripciones: Información de cuantas transcripciones tiene el usuario seleccionado.

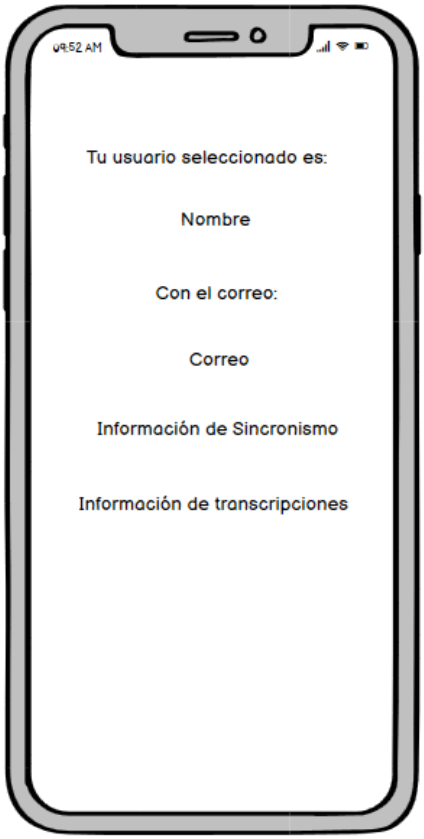


Figura 2.17. Escena Detalles de Usuario en rol Administrador

2.4.6. ESCENA "REPORTE GENERAL"

En la Figura 2.18. se indica la escena que se abre al seleccionar el *Text Reporte General* en el Menú Desplegable Administrador, en la cual se obtiene una información importante del proyecto. Esta escena está conformada por:

- Text usuarios Administradores: Texto de información de cantidad de usuarios Administradores.
- Text usuarios Familiares: Texto de información de cantidad de usuarios Familiares.
- Text usuarios Principales: Texto de información de cantidad de usuarios Principales.

- Text Transcripciones: Texto de información del total de transcripciones en la base de datos.
- Text Mensajes: Texto de información del total de mensajes en la base de datos.

Figura 2.18. Escena Reporte General

2.4.7. ESCENA "CREAR ADMINISTRADOR"

Se puede observar en la Figura 2.19. la escena que se abre al seleccionar el *Text Crear Administrador* en el Menú Desplegable Administrador, en la cual se puede crear un nuevo usuario Administrador. Esta escena está conformada por:

- Text Apodo o Nombre: Permite ingresar el apodo o nombre del usuario a crear.
- Text Email: Permite ingresar el correo.
- Text Clave: Permite ingresar la contraseña.
- Text Repetir Clave: Permite comprobar la contraseña ingresada.
- Botón Continuar: Permite registrar el usuario creado.
- Diálogo de Alerta: Permite confirmar el correo del usuario a crear.

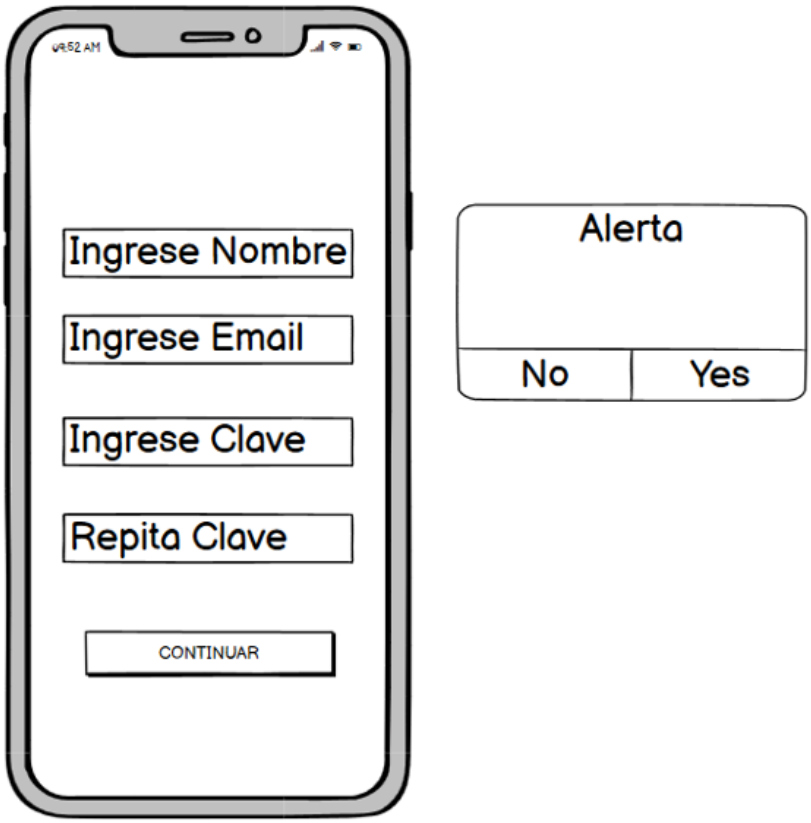


Figura 2.19. Escena Crear Administrador

Rol Principal

2.4.8. ESCENA "MENÚ DESPLEGABLE PRINCIPAL"

Como se indica en la Figura 2.20. esta escena se abre al seleccionar al botón *Ingresar* en el Inicio de Sesión y un correo perteneciente al rol Principal, en la cual se obtiene todas las acciones posibles para el rol Principal. Esta escena está conformada por:

- Imagen: Escudo de la Escuela Politécnica Nacional.
- Text Nombre: Nombre o apodo del usuario.
- Text Correo: Correo del usuario.
- Text Rol: Texto del rol al que pertenece el usuario.
- Text Sincronización Familiar: Texto que al seleccionar enviará a escena Sincronización Familiar.
- Text Eliminar Sincronismo: Texto que al seleccionar enviará a escena Eliminar Sincronismo.
- Text Transcripción: Texto que al seleccionar enviará a escena Transcripción.
- Text Ubicación: Texto que al seleccionar enviará a escena Ubicación.

- Text Portafolio: Texto que al seleccionar enviará a escena Portafolio.
- Text Reporte: Texto que al seleccionar enviará a escena Reporte General.
- Text Chat: Texto que al seleccionar enviará a escena Chat.
- Text Cerrar Sesión: Texto que al seleccionar cerrará sesión y enviará a escena Inicio de Sesión.

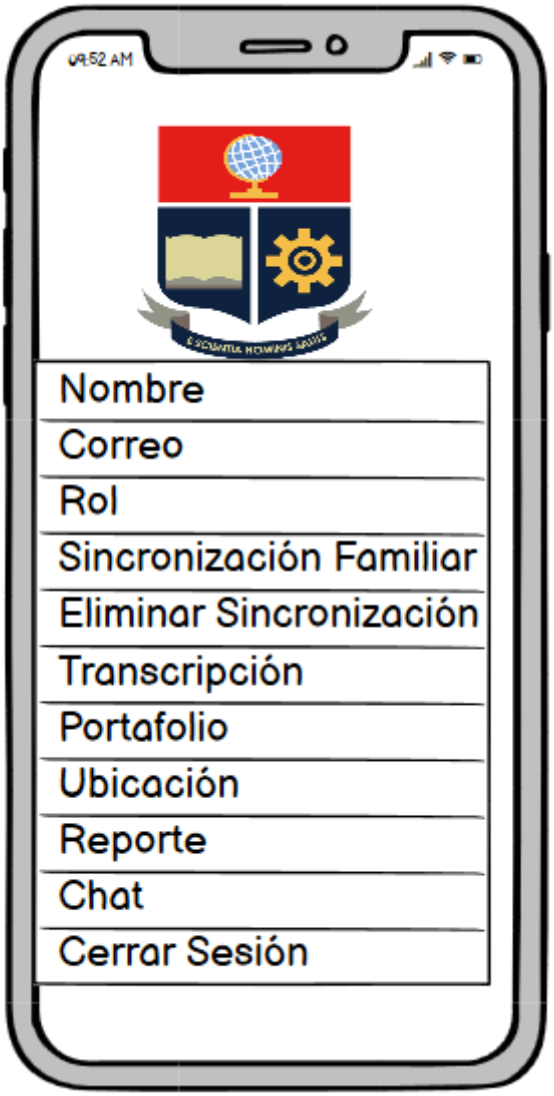


Figura 2.20. Escena Menú Desplegable Principal

2.4.9. ESCENA "SINCRONIZACIÓN FAMILIAR"

Como se muestra en la Figura 2.21. esta escena se abre al seleccionar el *Text Sincronización Familiar* en el Menú Desplegable Principal, en la cual se obtiene una lista de todos los usuarios Familiares. Esta escena está conformada por:

- Text Información: Informa que se debe seleccionar un usuario.
- Lista: Lista de usuarios Familiares.

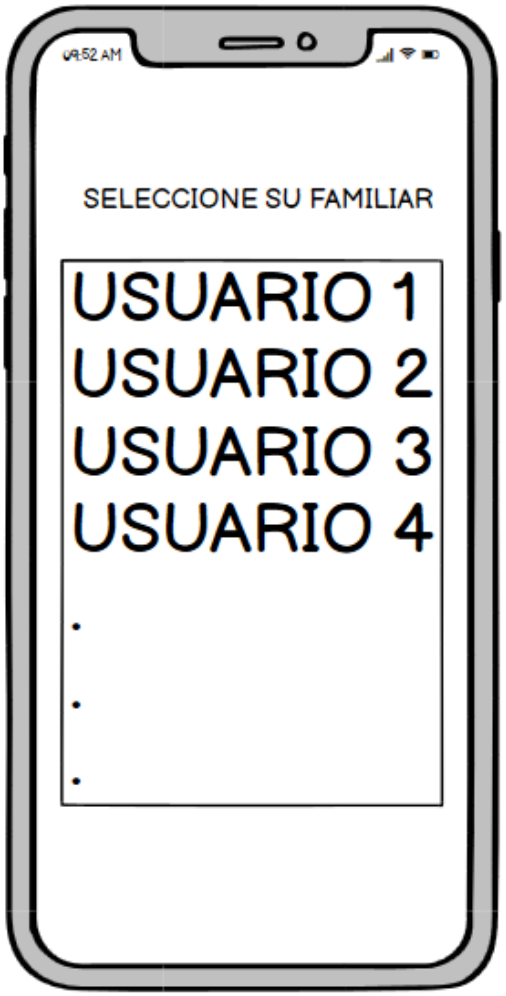


Figura 2.21. Escena Sincronización Familiar

2.4.10. ESCENA "ELIMINAR SINCRONIZACIÓN"

Como se muestra en la Figura 2.22. esta escena se abre al seleccionar el *Text Eliminar Sincronización* en el Menú Desplegable Principal, en la cual se obtiene información del usuario Familiar con el cual se tiene sincronización. Esta escena está conformada por:

- Text usuario: Texto de saludo a usuario Principal.
- Text usuario seleccionado: Texto usuario seleccionado.
- Text nombre usuario sincronismo: Texto donde se indica el nombre del usuario con el que se tiene sincronismo.
- Text correo: Texto correo.
- Text correo usuario sincronismo: Texto donde se indica el correo del usuario sincronismo.
- Text Alerta: Texto de alerta si se elimina sincronismo.
- Botón Eliminar Sincronismo: Se elimina sincronismo con el usuario Familiar.

- Botón Regresar: Regresa a escena Transcripción.

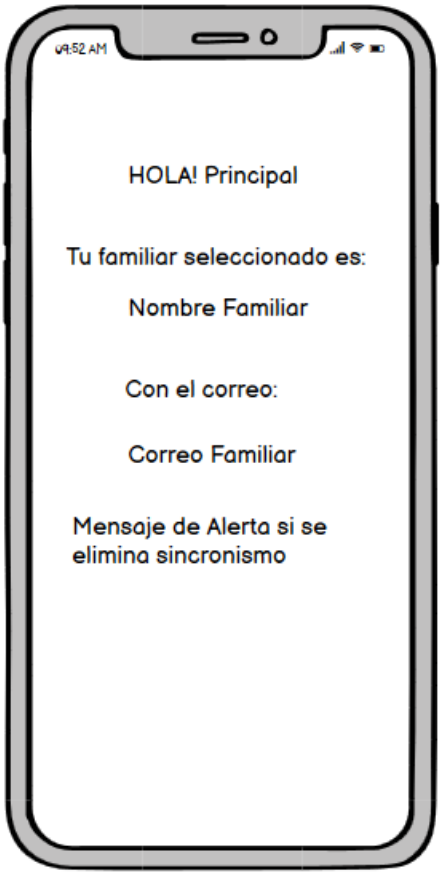


Figura 2.22. Escena Eliminar Sincronización

2.4.11. ESCENA "TRANSCRIPCIÓN"

Se puede observar en la Figura 2.23. la escena que se abre al seleccionar el *Text Transcripción* en el Menú Desplegable Principal, en la cual se obtiene la acción de transcribir. Esta escena está conformada por:

- Text Párrafo: Texto donde se podrá visualizar toda la transcripción finalizada.
- Botón Escuchar: Permite iniciar con la acción de transcribir.
- Botón Guardar: Permite guardar la transcripción.
- Botón Mas: Aumenta el tamaño de letra del Text Párrafo.
- Botón Menos: Disminuye el tamaño de letra del Text Párrafo.
- Edit Text Título: Este elemento permitirá ingresar un título para guardar transcripción.
- Diálogo de Alerta: Permite confirmar el título de la transcripción a guardar.
- Dialog Google: Permite Transcribir, pausar y enviar el texto de la transcripción a Text Párrafo.

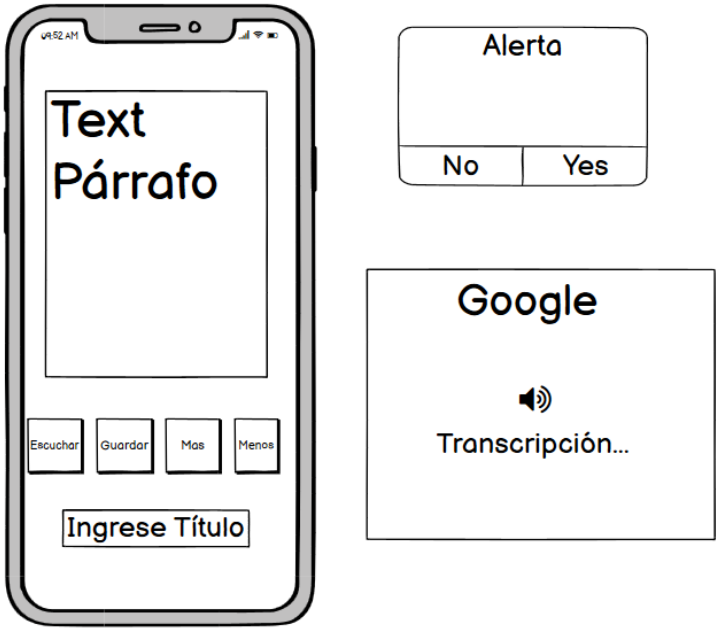


Figura 2.23. Escena Transcripción.

2.4.12. ESCENA "PORTAFOLIO"

Se puede observar en la Figura 2.24. la escena que se abre al seleccionar el *Text Portafolio* en el Menú Desplegable Principal, en la cual se obtiene una lista de todas las transcripciones guardadas que pertenecen al usuario. Esta escena está conformada por:

- Lista Transcripciones: Lista de transcripciones guardadas por el usuario.

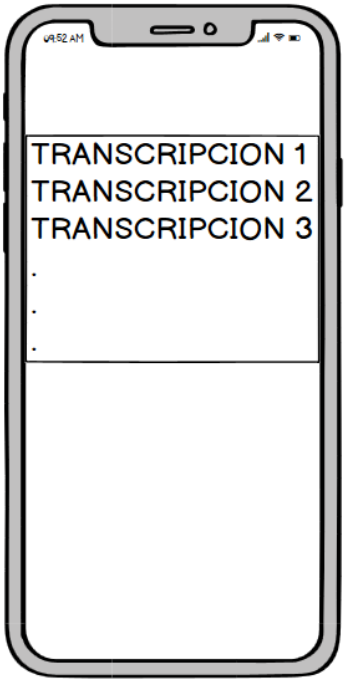


Figura 2.24. Escena Portafolio.

2.4.13. ESCENA “EDICIÓN DE TRANSCRIPCIÓN”

Como se observa en la Figura 2.25. esta escena se abre al seleccionar un elemento de la *Lista Portafolio* en la escena Portafolio, en la cual se puede editar, guardar y eliminar la transcripción seleccionada. Esta escena está conformada por:

- Text Título: Titulo de la transcripción seleccionada.
- Edit Text: Contenido de la transcripción que se puede editar.
- Botón Guardar: Elemento que guardará la edición.
- Botón Eliminar: Elemento que eliminará la transcripción.
- Botón Regresar: Elemento que regresará a escena Portafolio.

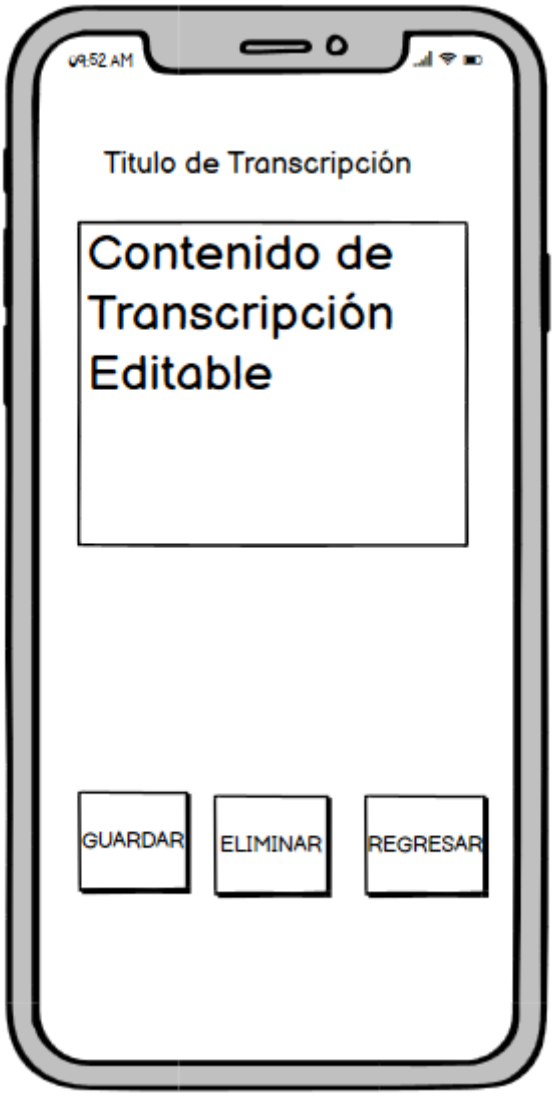


Figura 2.25. Escena Edición de Transcripción.

2.4.14. ESCENA “UBICACIÓN”

En la Figura 2.26. se indica la escena que se abre al seleccionar *Text Ubicación* en el Menú Desplegable Principal, en la cual se puede ver la ubicación actual y enviarla a un contacto de WhatsApp. Esta escena está conformada por:

- Mapa: Indica la ubicación actual y ubicaciones previas.
- Botón Enviar Ubicación: Elemento que permitirá sincronizar con WhatsApp y enviará la ubicación actual a un contacto.

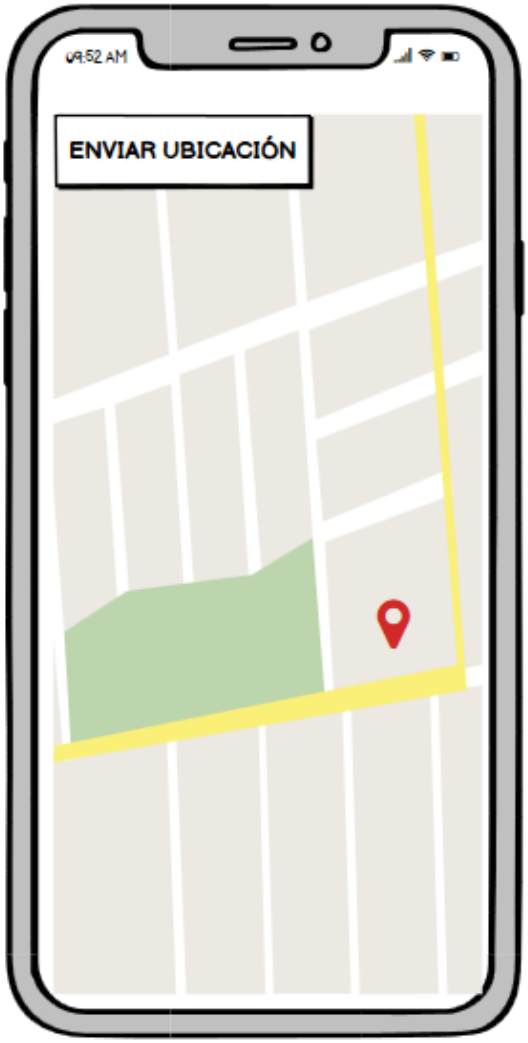


Figura 2.26. Escena Ubicación

2.4.15. ESCENA “REPORTE”

Como se muestra en la Figura 2.27. esta escena se abre al seleccionar *Text Reporte* en el Menú Desplegable Principal, en la cual se puede ver una lista de transcripciones y descargar un archivo PDF con la información. Esta escena está conformada por:

- Botón *Guardad* Reporte: Permite guardar un reporte en PDF en el dispositivo.
- Lista Transcripciones: Permite visualizar la lista de todas las transcripciones del usuario por nombre y fecha,

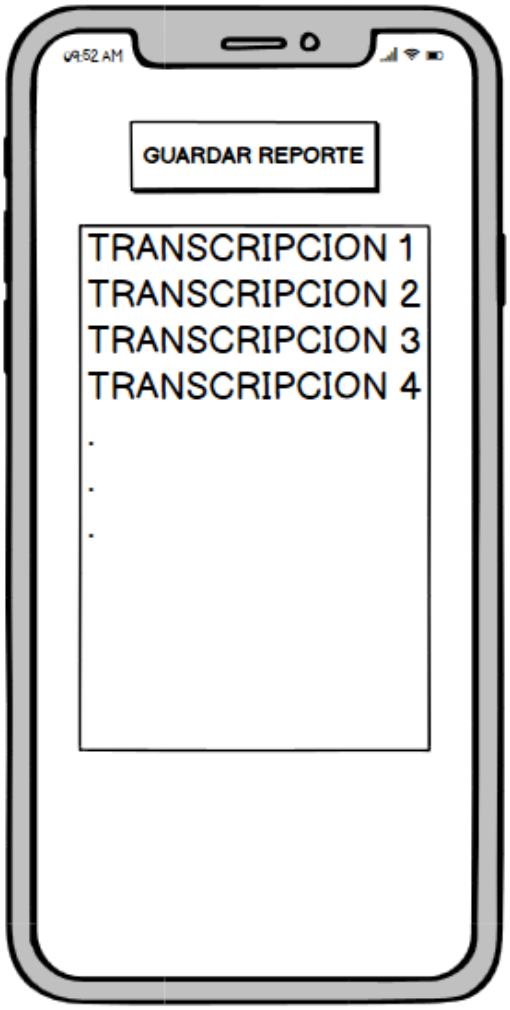


Figura 2.27. Escena Reporte

2.4.16. ESCENA "CHAT"

Como se puede ver en la Figura 2.28. esta escena se abre al seleccionar *Text Chat* en el Menú Desplegable Principal, en la cual se puede ver una lista de mensajes y enviar nuevos mensajes. Esta escena está conformada por:

- Text Mensajes: Texto Mensajes.
- Lista Mensajes: Mensajes que se tiene en ese momento.
- Edit Text Texto: Espacio donde se escribe el mensaje a enviar.
- Botón Enviar: Permite enviar el mensaje escrito.

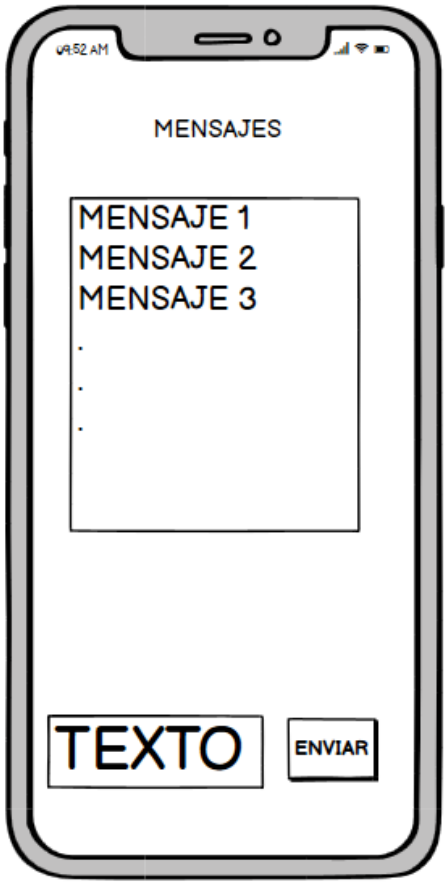


Figura 2.28. Escena Chat

A continuación, se indica las escenas para el *Rol Familiar*

2.4.17. ESCENA "MENÚ DESPLEGABLE FAMILIAR"

En la Figura 2.29. se observa la escena que se abre al seleccionar al botón *Ingresar* en el Inicio de Sesión y un correo perteneciente al rol Familiar, en la cual se obtiene todas las acciones posibles para el rol Familiar. Esta escena está conformada por:

- Imagen: Escudo de la Escuela Politécnica Nacional.
- Text Nombre: Nombre o apodo del usuario.
- Text Correo: Correo del usuario.
- Text Rol: Texto del rol al que pertenece el usuario:
- Text Ver Ubicación: Texto que al seleccionar enviará a escena Ver Ubicación.
- Text Reporte: Texto que al seleccionar enviará a escena Reporte.
- Text Chat: Texto que al seleccionar enviará a escena Chat.
- Text Cerrar Sesión: Texto que al seleccionar cerrará sesión y enviará a escena Inicio de Sesión.

Figura 2.29. Escena Menú Desplegable Familiar

2.4.18. ESCENA “VER UBICACIÓN”

Como se muestra en la Figura 2.30. esta escena se abre al seleccionar *Text Ver Ubicación* en el Menú Desplegable Familiar, en la cual se puede ver ubicaciones con fecha del usuario Principal sincronizado. Esta escena está conformada por:

- Mapa: Indica la ubicación actual.

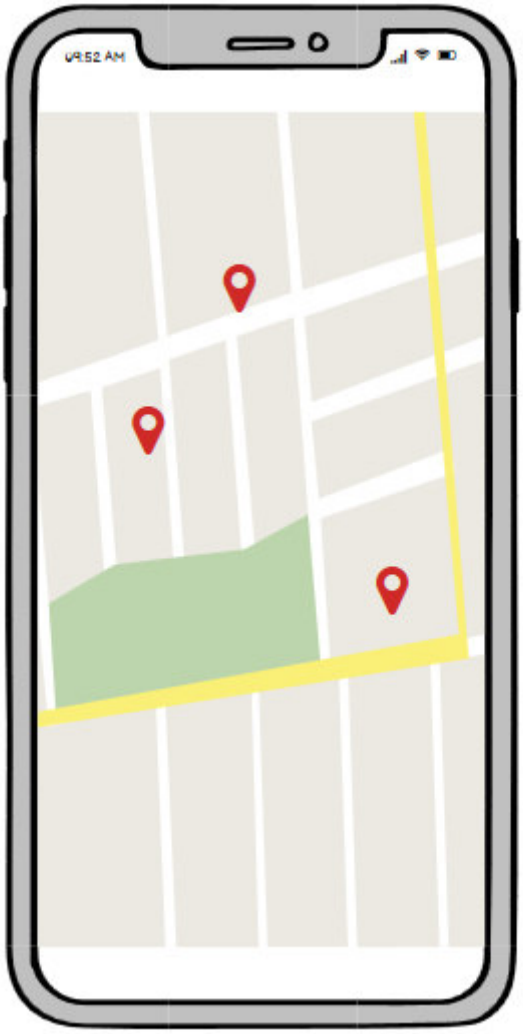


Figura 2.30. Escena Ver Ubicación

2.4.19. ESCENA “REPORTE”

Como se puede ver en la Figura 2.31. esta escena se abre al seleccionar *Text Reporte* en el Menú Desplegable Familiar, en la cual se puede ver una lista de transcripciones del usuario Principal sincronizado y descargar un archivo PDF con la información. Esta escena está conformada por:

- Botón Guardad Reporte: Permite guardar un reporte en PDF en el dispositivo.
- Lista Transcripciones: Permite visualizar la lista de todas las transcripciones del usuario.

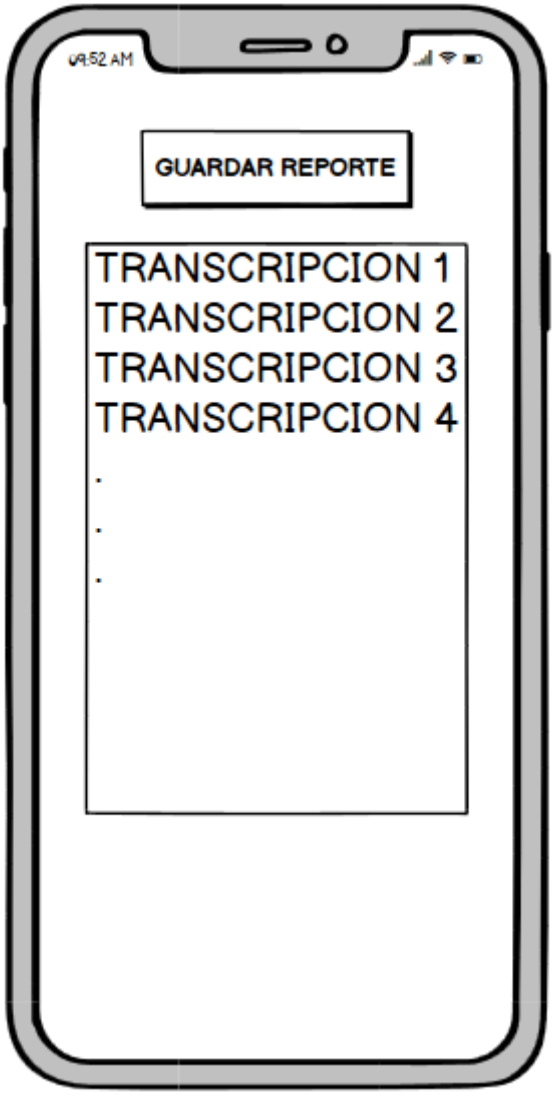


Figura 2.31. Escena Reporte

2.4.20. ESCENA "CHAT"

En la Figura 2.32. se indica la escena que se abre al seleccionar *Text Chat* en el Menú Desplegable Familiar, en la cual se puede ver una lista de mensajes y enviar nuevos mensajes. Esta escena está conformada por:

- Text Mensajes: Texto Mensajes.
- Lista Mensajes: Mensajes que se tiene en ese momento.
- Edit Text Texto: Espacio donde se escribe el mensaje a enviar.
- Botón Enviar: Permite enviar el mensaje escrito.

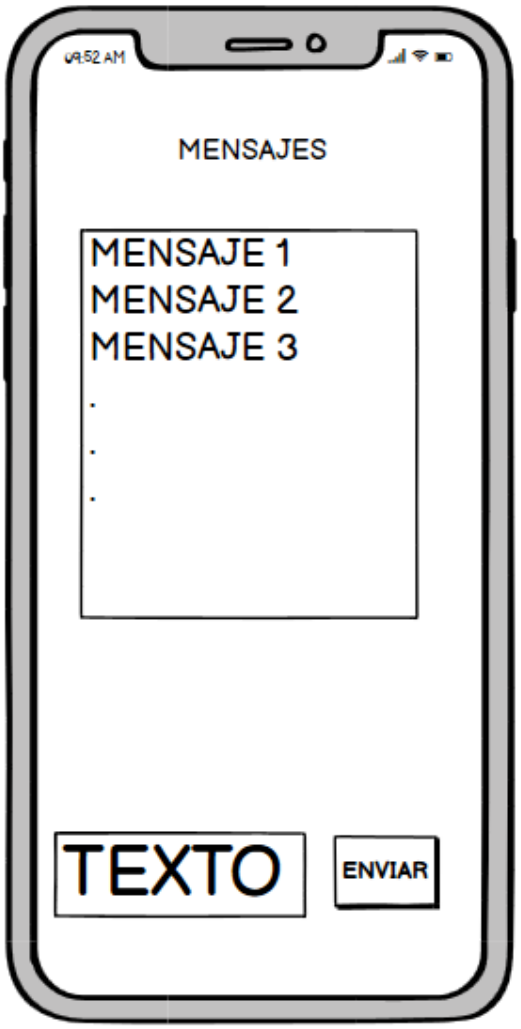


Figura 2.32. Escena Chat

2.5. IMPLEMENTACIÓN DEL PROTOTIPO

Para iniciar con la implementación de la aplicación es necesario tener instalado la aplicación de Android Studio en la computadora a utilizar, esta aplicación se la puede descargar del sitio oficial de Android Studio y su instalación es fácil de hacerla. Hay que recordar que el prototipo se desarrollará en lenguaje de programación Java.

2.5.1. CREACIÓN DEL PROYECTO EN FIREBASE

Es necesario tener la sincronía entre el proyecto de Android Studio con Firebase, para eso es necesario obtener una cuenta de Google que permitirá el ingreso a la plataforma como se observa en la Figura 2.33.

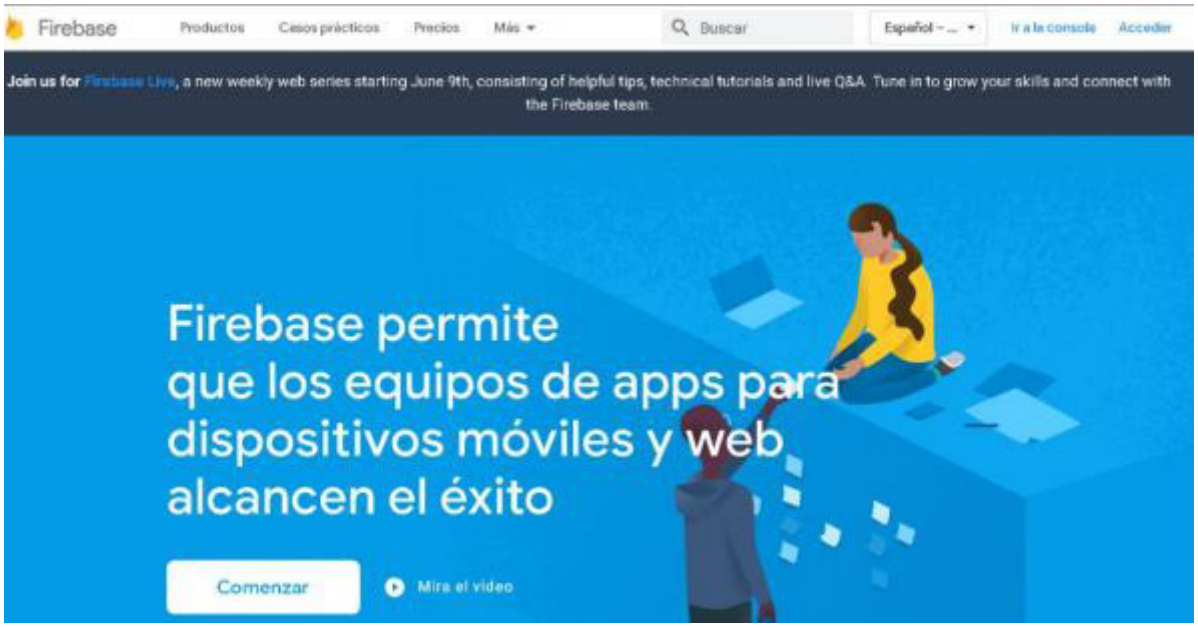


Figura 2.33. Página de Firebase

Luego de cumplir con la autenticación en la cuenta de Firebase se pulsa el botón *consola* en donde se puede visualizar proyectos creados o crear un proyecto nuevo (Figura 2.34).

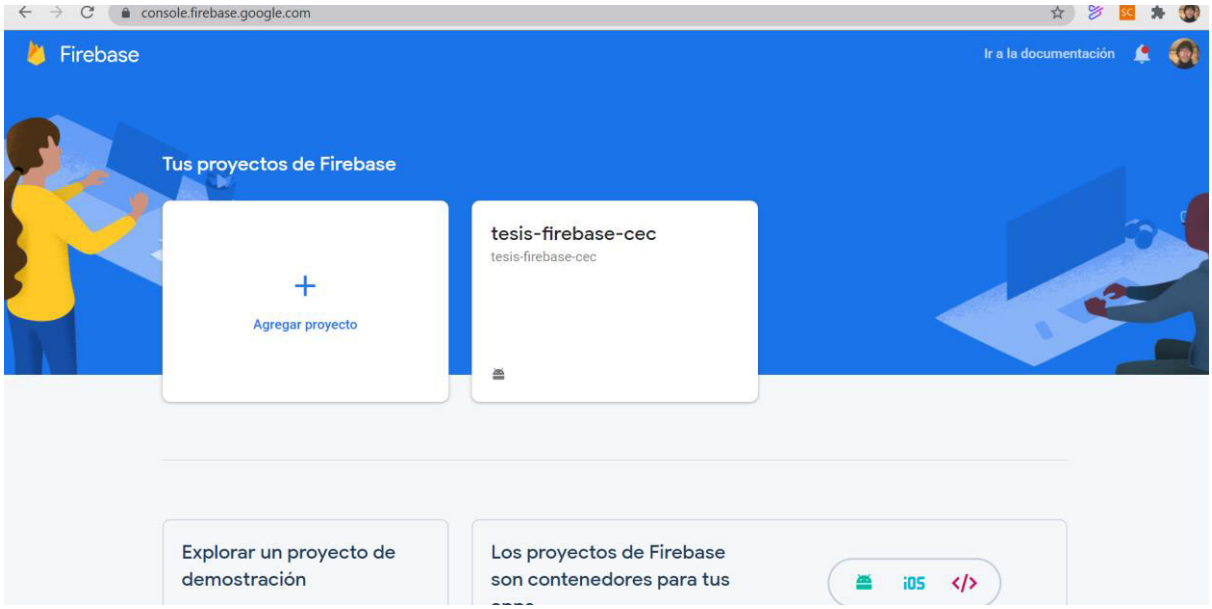


Figura 2.34. Proyectos de Firebase

Si se da clic en *Agregar proyecto* es posible crear un nuevo proyecto donde se debe seguir los pasos indicados por la misma plataforma de Firebase como se indica en la Figura 2.35.

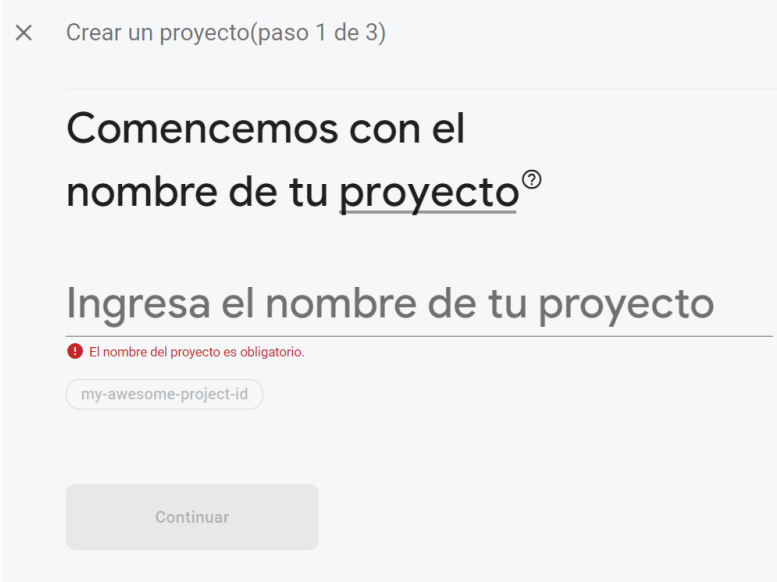


Figura 2.35. Nombre del proyecto a crear

Luego se activa Google Analytics la cual permite realizar predicciones, informes o segmentaciones de usuarios, es importante que se active ya que mejora el rendimiento y eficiencia de Firebase (Figura 2.36).

Figura 2.36. Google Analytics

Para finalizar se toma la cuenta por *Default* de Google Analytics donde se guardarán todos los datos durante el funcionamiento y utilidad de la aplicación como se observa en la Figura 2.37.

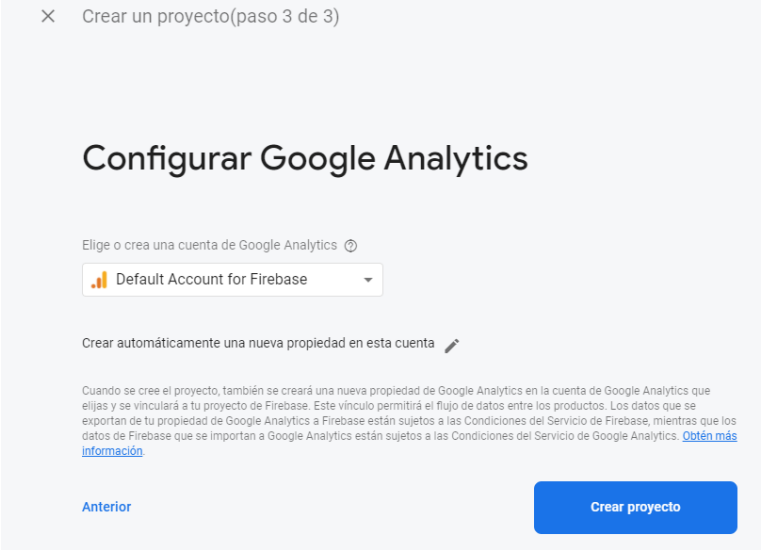


Figura 2.37. Selección de cuenta Google Analytics

En la pantalla principal de la consola de Firebase se puede observar en la parte izquierda todas las funcionalidades como son Authentication, Firestore, Realtime Database entre otras. En la parte central se puede seleccionar la plataforma en donde se desea implementar como se indica en la Figura 2.38.

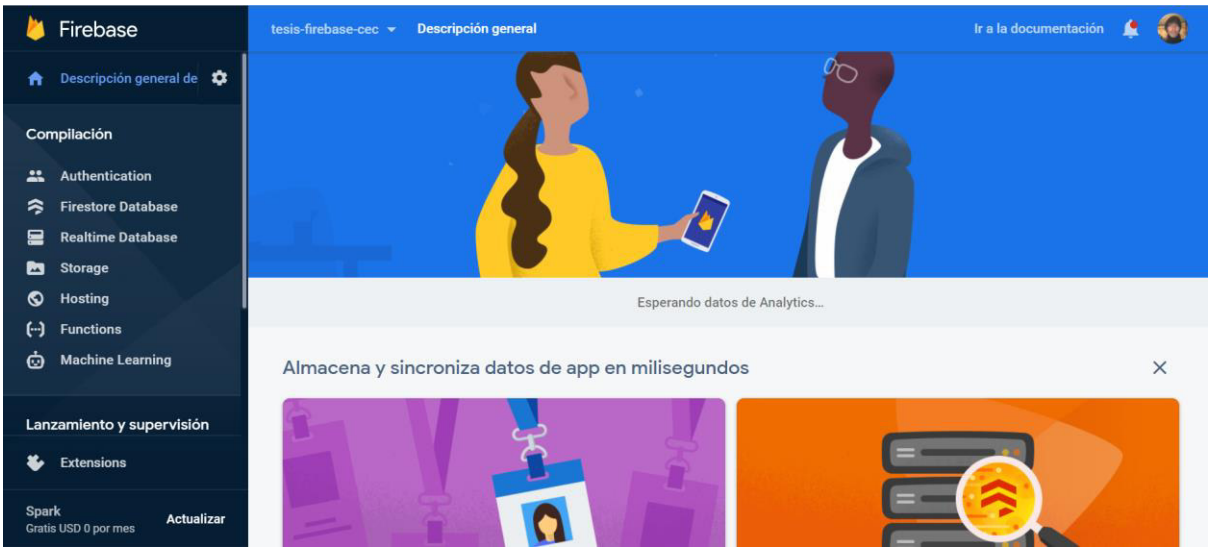


Figura 2.38. Consola para un proyecto en Firebase

Al seleccionar el ícono de Android, se mostrará una nueva interfaz en la cual se registrará la aplicación para tener sincronía con el proyecto de Android Studio, una parte importante es el certificado de firma SHA-1, esta información se encuentra en el archivo build.grade (Module:app) del proyecto en Android Studio (Figura 2.39).

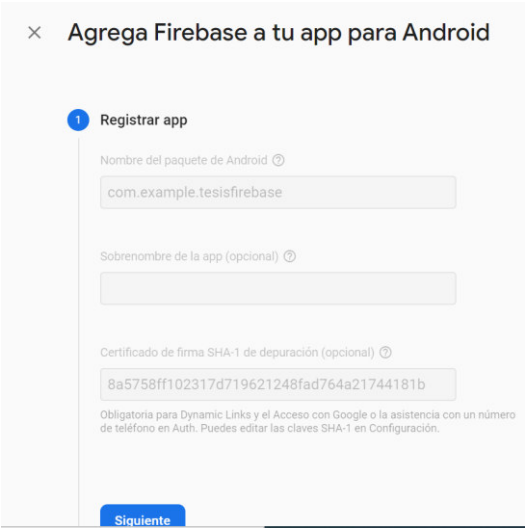


Figura 2.39. Proceso inicial para agregar Firebase a una app

Una vez la información solicitada sea registrada, automáticamente se generará un archivo .json que deberá ser descargado y copiado en el proyecto de Android Studio como se puede ver en la Figura 2.40.

Figura 2.40. Archivo de configuración. Json

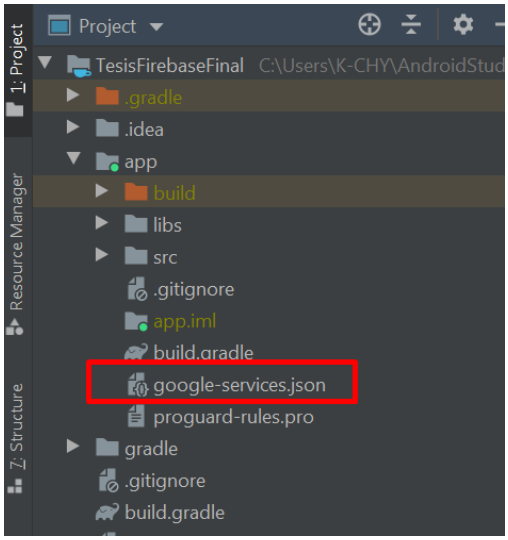


Figura 2.41. Archivo de configuración. Json en Android Studio

Luego se debe añadir el SDK de Firebase a la aplicación, para poder usar el complemento se debe modificar el archivo build.gradle a nivel de proyecto y se agrega las líneas de código como se observa en la Figura 2.42.

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.5'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

Figura 2.42. Dependencias a nivel de proyecto

También se debe modificar el archivo build.gradle a nivel de aplicación y se agrega las líneas de código como se indica en la Figura 2.43.

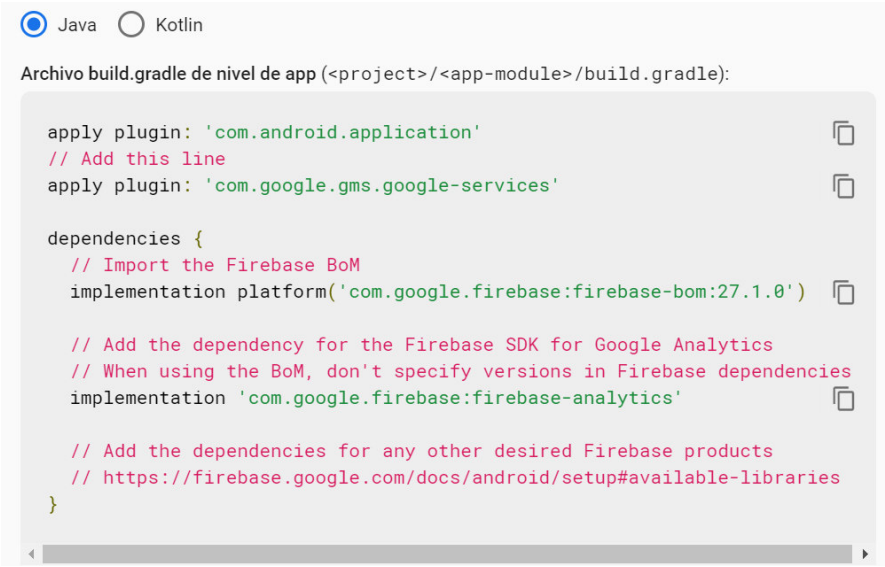


Figura 2.43. Dependencias a nivel de aplicación

Para finalizar y poner en sincronía la aplicación y Firebase se debe dar clic en *Sync Now,* que está ubicado en la parte superior derecha del archivo *build.grade.*

2.5.2. AUTENTICACIÓN DE USUARIOS FIREBASE

Una de las partes del proyecto (Figura 2.44.) es la autenticación de usuarios, para ello es necesario activar *Firebase Authentication* la cual permite varias opciones de autenticación como son Google, Facebook, Twitter y la que se implementará en el proyecto que es por correo electrónico.

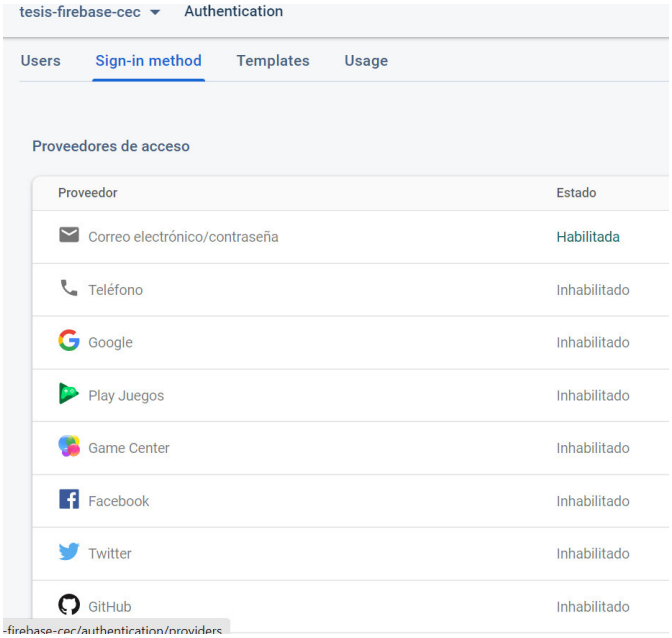


Figura 2.44. Proveedores para autenticación en Firebase

Para el proyecto se activará autenticación por medio de correo y contraseña como se observa en la Figura 2.45.

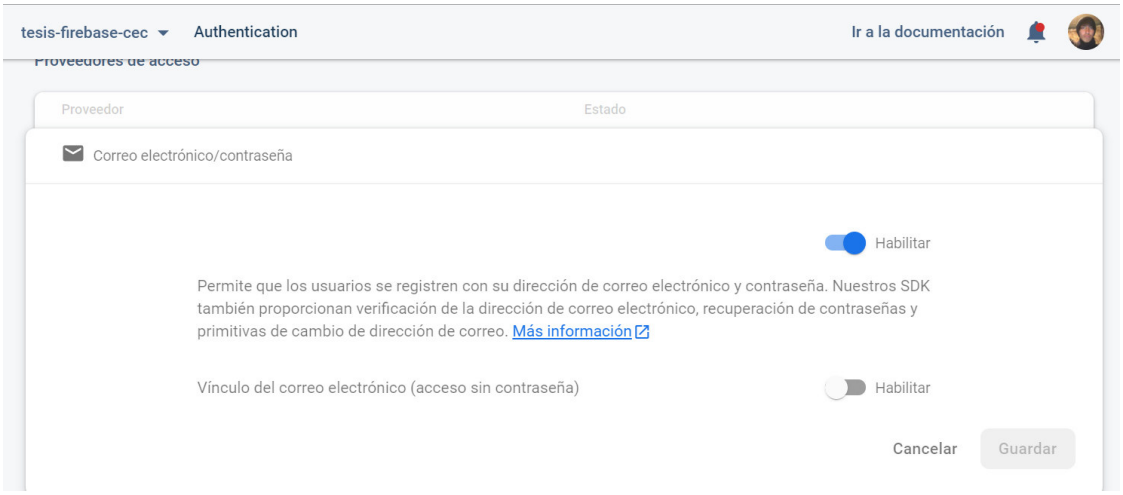


Figura 2.45. Servicio de autenticación habilitado

Para tener sincronía con la aplicación se debe agregar la biblioteca en *build.gradle* a nivel de aplicación con el siguiente código y sincronizar desde Android Studio como se indica en la Figura 2.46.

```
implementation 'com.google.firebase:firebase-auth:19.2.0'
```

Figura 2.46. Implementación de autenticación en aplicación

2.5.3. BASE DE DATOS REALTIME DE FIREBASE

Realtime Database servirá para almacenar toda la información y datos de la aplicación, para activar el servicio se debe ir a la parte izquierda del menú principal de Firebase y crear base de datos (Figura 2.47).

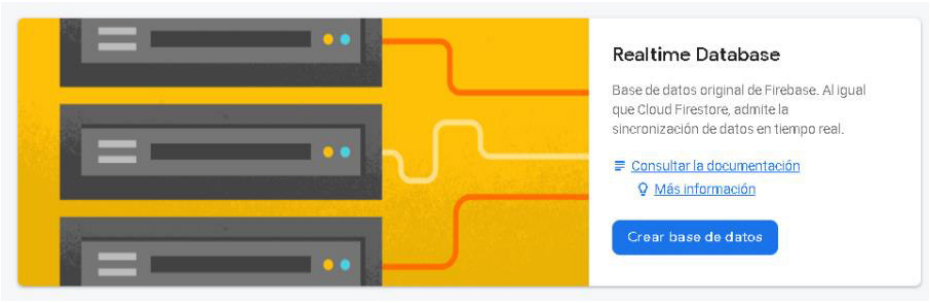


Figura 2.47. Crear base de datos en Firebase

Aparecerá una nueva interfaz la cual son reglas de seguridad para la base de datos, en esta opción se selecciona la segunda opción y se habilita como se observa en la Figura 2.48.

Figura 2.48. Reglas de seguridad en Realtime

Al momento de habilitar se creará la base de datos, para el proyecto se crea de manera inicial tres nodos: Mensajes, Transcripciones y Usuarios como se observa en la Figura 2.49.

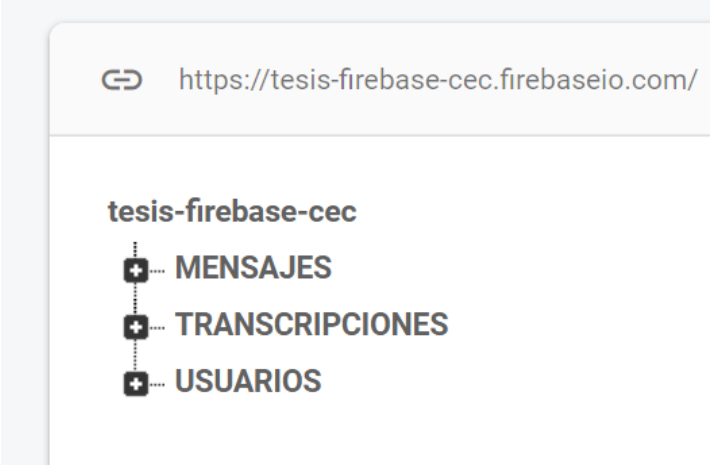


Figura 2.49. Nodos Base

Adicionalmente se debe activar la biblioteca en el proyecto dentro de Android Studio con el siguiente código y sincronizando como se observa en la Figura 2.50.

```
implementation 'com.google.firebase:firebase-database:19.4.0'
```

Figura 2.50. Implementación de real time en aplicación

2.5.4. IMPLEMENTACIÓN DE APIS DE GOOGLE

Las APIs de Google son fundamentales para el desarrollo de la aplicación, para la activación de esta plataforma es necesario tener una cuenta el desarrollador de Google y acceder a este.

Cuando se ingresa por primera vez se debe aceptar los términos y condiciones, y seleccionar *Crear Proyecto* como se indica en la Figura 2.51.

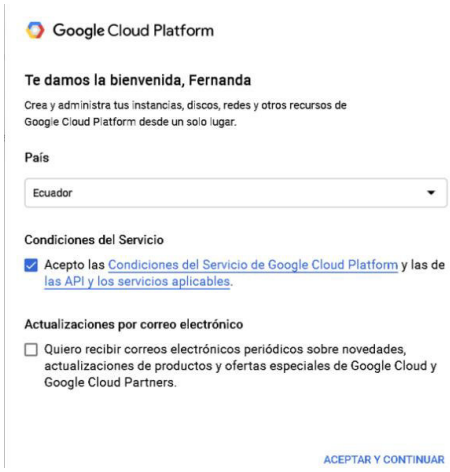


Figura 2.51. Consola de Desarrollo de Google

Se necesita ingresar un nombre para el proyecto y una ubicación, esta última se puede seleccionar “Sin organización” (Figura 2.52).

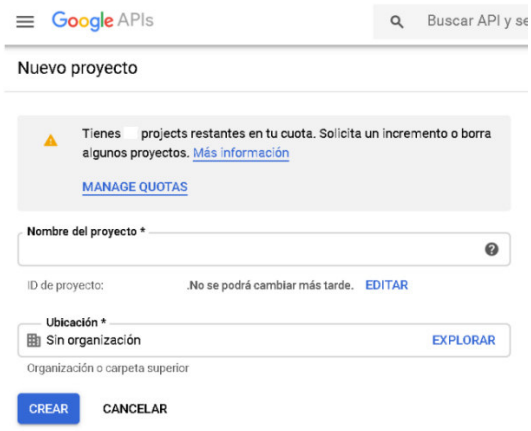


Figura 2.52. Creación de proyecto

Al final se puede ver la biblioteca de API donde se podrá agregar las diferentes APIs a utilizar en la aplicación como se observa en la Figura 2.53.

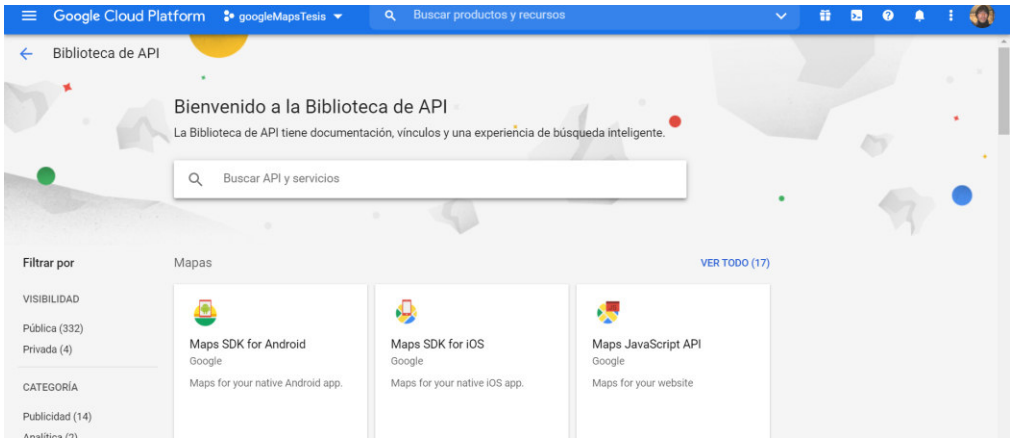


Figura 2.53. Biblioteca de APIs

2.5.5. IMPLEMENTACIÓN DE GOOGLE MAPS API

Google Maps es una herramienta importante en el diseño de mapas, gracias a esto existen diferentes vistas de cada creador en cada aplicación. Es necesario crear una credencial que ayudará a tener una sincronía con el proyecto en Android Studio como se observa en la Figura 2.54.

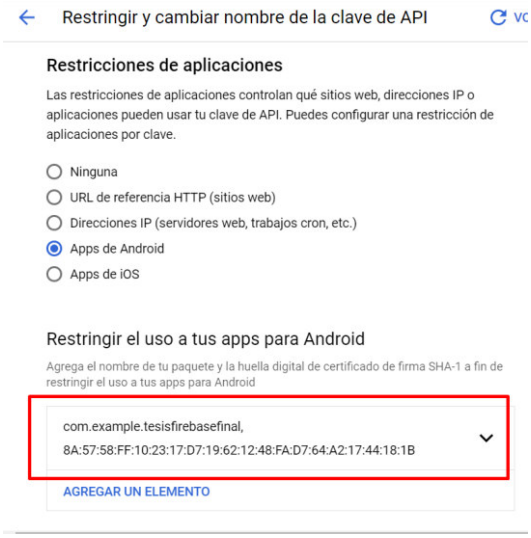


Figura 2.54. Credencial de API de Google

Es necesario crear una Actividad que se encargue del procesamiento de mapas dentro de la aplicación, para ello Android Studio facilita con una Activity dedicada a este trabajo la cual se la puede encontrar como se indica en la Figura 2.55.

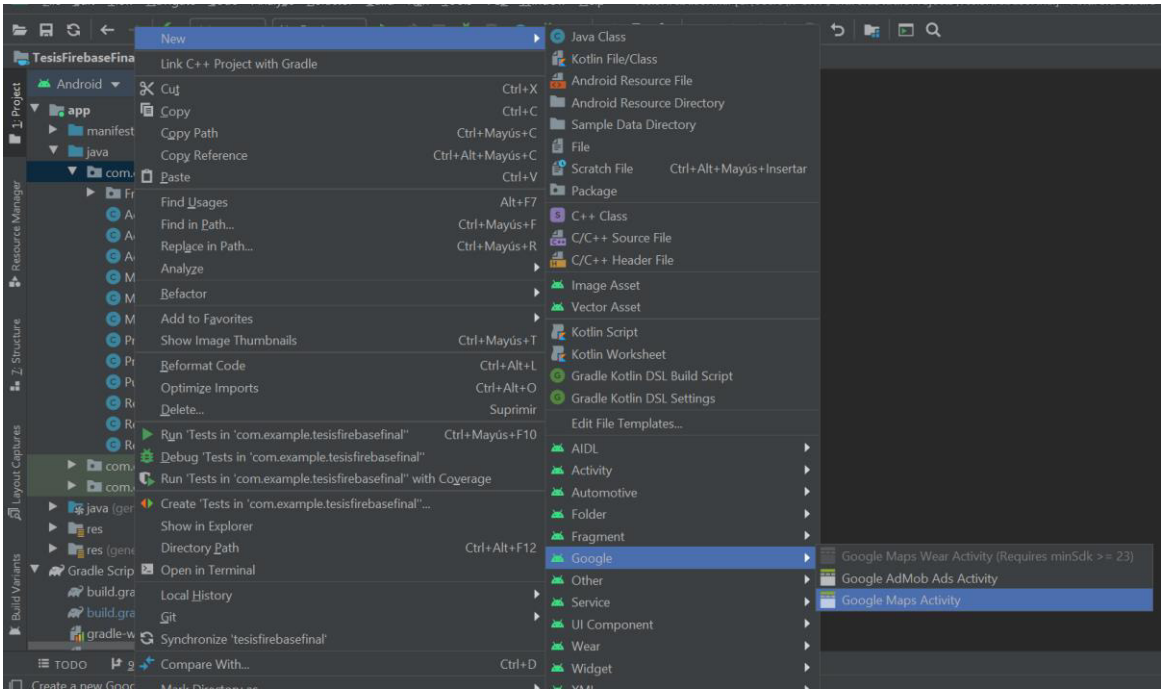


Figura 2.55. Actividad de soporte de Google Maps en Android Studio

Además, se debe sincronizar desde el proyecto en Android Studio hacia Google Maps, para esto se dirige a la carpeta *values* del módulo *res* y en *google_maps_api.xml* se copia la credencial antes creada (Figura 2.56).

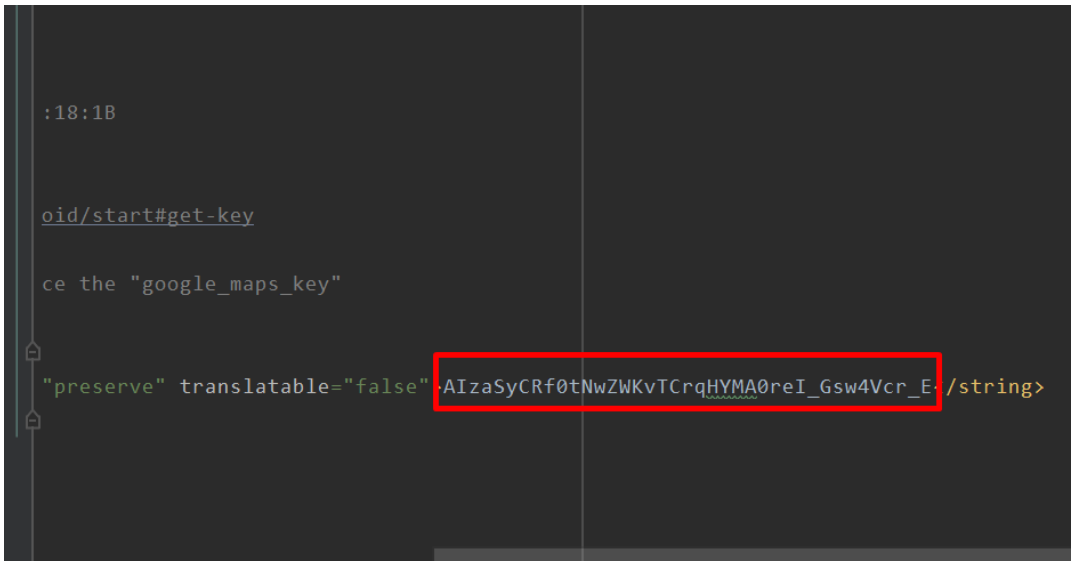


Figura 2.56. Sincronía entre Google Maps y Android Studio

Para finalizar se debe instalar las dependencias y bibliotecas con los siguientes códigos en *build.gradle* a nivel de aplicación como se observa en la Figura 2.57 y Figura 2.58.

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

Figura 2.57. Implementación de Google Maps en aplicación

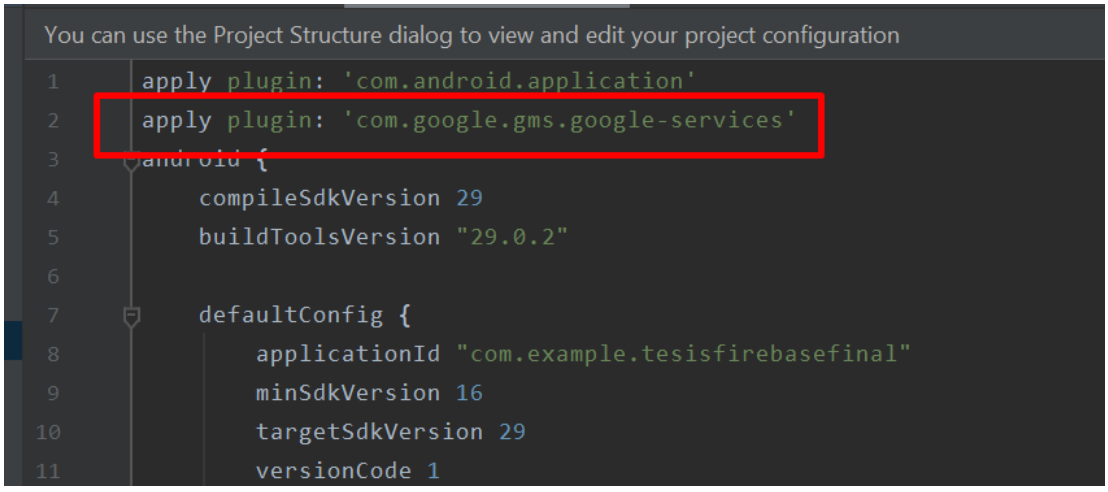


Figura 2.58. Dependencia para soporte de servicio de direcciones

2.5.6. ESTRUCTURA DEL PROYECTO EN ANDROID STUDIO

En las Figuras 2.59, 2.60 y 2.61 se pueden ver los diferentes elementos que conforman la estructura de la aplicación respecto a Activity y Fragment, se analizará todos los campos necesarios para el correcto funcionamiento de la aplicación con su explicación.

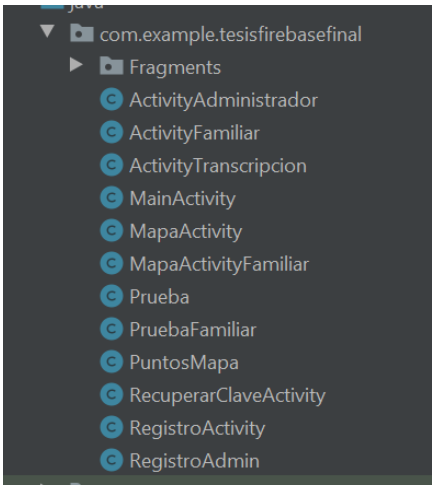


Figura 2.59. Estructura de Activity

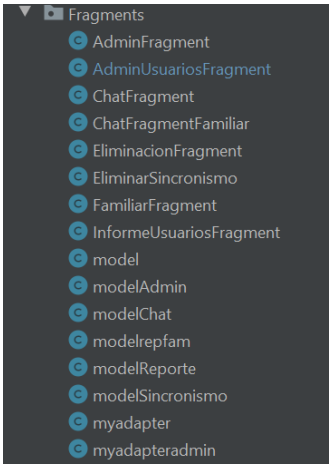


Figura 2.60. Estructura de Fragment

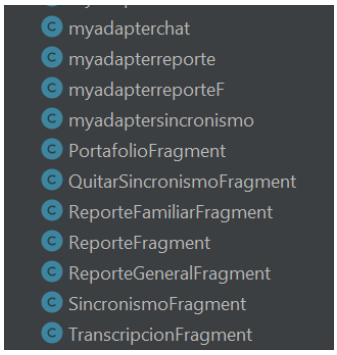


Figura 2.61. Continuación de estructura de Fragment

2.5.7. ACTIVIDAD DE REGISTRO (REGISTROACTIVITY)

Es la actividad para el registro de usuarios dentro de la aplicación, se pedirá varios requisitos para poder ser completada y se plantea la creación del usuario perteneciente al rol Familiar y al rol Principal.

A continuación, se detalla las líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, variables como “claveS” que se utilizará para procesos de asignación de caracteres que varían y atributos pertenecientes a la base de datos como se observa en el código 2.1.

```
public class RegistroActivity extends AppCompatActivity {
    private Button buttonContinuar;
    private EditText clave,claveRep,apodo,email;
    private FirebaseAuth baseAutenticacion;
    private RadioButton rbMAprincipal,rbMAfamiliar;
    String claveS;
    String claveRepS;
    String emailS;
    String sincronismo="0";
    String apodoS,claseUsuario;
    private DatabaseReference baseDatos;
```

Código 2.1. Creación de variables RegistroActivity

- **Código para inicialización de elementos**

Se inicializa dos radioButton para seleccionar a que rol pertenecerá el usuario a crear, cuatro editText para nombre, email, clave y confirmación de clave; un botón para continuar con el registro del usuario y los dos campos para autenticación y base de datos de Firebase como se mira en el código 2.2.

```
clave=(EditText) findViewById(R.id.editTextClaveRA);
claveRep=(EditText) findViewById(R.id.editTextClaveRepRA);
rbMAprincipal=(RadioButton)findViewById(R.id.radioButtonPrincipal);
rbMAfamiliar=(RadioButton)findViewById(R.id.radioButtonFamiliar);
apodo=(EditText)findViewById(R.id.editTextApodoRA);
email=(EditText)findViewById(R.id.editTextEmailRA);
buttonContinuar=(Button) findViewById(R.id.btnContinuar);
baseAutenticacion= FirebaseAuth.getInstance();
// apuntamos hacia base de datos
baseDatos= FirebaseDatabase.getInstance().getReference();
```

Código 2.2. Inicialización de elementos en RegistroActivity

- **Código botón Continuar**

Al seleccionar el botón Continuar se captura los datos ingresados y se los almacena en variables de tipo string, se comprueba que los campos no están vacíos, luego que las contraseñas coincidan y que se haya seleccionado el tipo de rol a crear como se indica en el código 2.3.

```
public void btnContinuar (){
    claveS=clave.getText().toString();
    claveRepS=claveRep.getText().toString();
    emailS=email.getText().toString();
    apodoS=apodo.getText().toString();
    //aseguramos que no este vacio ningun campo
    if(!claveRepS.isEmpty()&&!claveS.isEmpty()&&!apodoS.isEmpty()&&!emailS.isEmpty()){
        //aseguramos que las claves ingresadas sean las mismas
        if (claveS.equals(claveRepS)){
            if (claveS.length()>5){
                if(rbMAfamiliar.isChecked()==false&&rbMAprincipal.isChecked()==false){
                    Toast.makeText( context: this,  text: "Seleccione su Usuario", Toast.LENGTH_SHORT).show();
                }else {
                    //validacion de principal o familiar
                    if (rbMAprincipal.isChecked() == true) {
                        claseUsuario = "PRINCIPAL";
                    } else if (rbMAfamiliar.isChecked() == true) {
                        claseUsuario = "FAMILIAR";
                    }
                    //
```

Código 2.3. Código botón Continuar en RegistroActivity

- **Alerta de** diálogo **botón Continuar**

Se crea un diálogo de alerta para confirmar que el correo ingresado anteriormente es el correcto, al seleccionar *SI* pasamos a la función *registrarUsuario()* y al seleccionar *NO* se mantiene en el Activity para hacer un cambio a cualquier ítem (código 2.4).

```
//
final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder(
        context: RegistroActivity.this);
confirmacionDialog.setMessage(Html.fromHtml( source: "Confirma que el correo es: "
        + "<b>" + emailS + "</b>")).setCancelable(false)
        .setPositiveButton( text: "Si", (dialog, which) -> {
                registrarUsuario();
        }).setNegativeButton( text: "No", (dialog, which) -> {
        dialog.cancel();
});
AlertDialog alerta = confirmacionDialog.create();
alerta.setTitle("Cofirmación");
alerta.show();
```

Código 2.4. Alerta de diálogo botón Continuar en RegistroActivity

- **Código para función registrarUsuario**

En esta función se llama a *createUserWithEmailAndPassword(emailS,claveS)* de autenticación en Firebase la cual pide un email y una clave para crear el usuario y registrarlo directamente en el módulo de autenticación en Firebase, además, se apunta a Database Realtime de Firebase, se crea un mapa de atributos *Map<String,Object>* el cual acepta 5 objetos para más tarde guardar en el nodo perteneciente al usuario dentro de la base de datos; también se envía un correo de confirmación de cuenta al email perteneciente al usuario creado como se observa en el código 2.5.

```
public void registrarUsuario(){

    baseAutenticacion.createUserWithEmailAndPassword(emailS,claveS).addOnCompleteListener((task) → {
            if (task.isSuccessful()){
                //baseAutenticacion.setLanguageCode("es");
                String id=baseAutenticacion.getCurrentUser().getUid();
                Map<String,Object> mapUsuario=new HashMap<>();
                mapUsuario.put( k: "idPropio",id);
                mapUsuario.put( k: "Apodo",apodoS);
                mapUsuario.put( k: "Correo",emailS);
                mapUsuario.put( k: "Clave",claveS);
                mapUsuario.put( k: "Sincronismo",sincronismo);
                baseDatos.child("USUARIOS").child(claseUsuario).child(id).setValue(mapUsuario).addOnCompleteListener((task2) → {
                        if (task2.isSuccessful()){
                            //ir a activity que ya transcribe
                            FirebaseUser UserBD=baseAutenticacion.getCurrentUser();
                            UserBD.sendEmailVerification();
                            Toast.makeText( context: RegistroActivity.this,  text: "Usuario creado," +
                                    " confirme en su correo por favor", Toast.LENGTH_SHORT).show();
                            finish();
                        }else{
                            Toast.makeText( context: RegistroActivity.this,  text: "Por el momento no es " +
                                    "posible crear usuarios, disculpas", Toast.LENGTH_SHORT).show();
                        }
                });
            }else{
                Toast.makeText( context: RegistroActivity.this,  text: "El email ya existe", Toast.LENGTH_SHORT).show();
```

Código 2.5. Creación de variables RegistroActivity

2.5.8. ACTIVIDAD PARA RECUPERAR CLAVE (RECUPERARCLAVEACTIVITY)

Es la actividad para recuperar o cambiar la clave de un usuario dentro de la aplicación, se pedirá varios requisitos para poder ser completada y se plantea el envío de un mensaje al correo ingresado para su cambio de clave.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, variables como “emailS” que se utilizará para procesos de asignación de caracteres que varían, barras de progreso y atributos pertenecientes a la base de datos como se indica en el código 2.6.

```
public class RecuperarClaveActivity extends AppCompatActivity {

    private EditText correoRCA;
    private Button buttonConfirmar;
    private String emailS;
    private FirebaseAuth authCambio;
    private ProgressDialog progreso;
```

Código 2.6. Código para RecuperarActivity

- **Código para inicialización de elementos**

Se inicializa un editText para el email al cual se enviará el correo para el cambio de clave, un botón para confirmar, el campo para autenticación de Firebase y el objeto para mostrar el progreso como se puede ver en el código 2.7.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recuperar_clave);
    correoRCA=(EditText)findViewById(R.id.editTextEmailRecuperar);
    buttonConfirmar=(Button)findViewById(R.id.btnEnviar);
    authCambio=FirebaseAuth.getInstance();
    progreso=new ProgressDialog( context: this);
```

Código 2.7. Inicialización de elementos en RecuperarActivity

- **Código botón Confirmar**

Se captura el string del correo ingresado, se inicializa el progreso y si el campo email no está vacío se llama a la función *cambiarClave()* como se observa en el código 2.8.

```
    buttonConfirmar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            emailS=correoRCA.getText().toString();
            if(!emailS.isEmpty())
            {
                progreso.setMessage("Enviando mensaje");
                progreso.setCanceledOnTouchOutside(false);
                progreso.show();
                cambiarClave();
            }else {
                Toast.makeText( context: RecuperarClaveActivity.this,  text: "Ingrese Correo", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Código 2.8. Botón Confirmar en RecuperarActivity

- Código cambiarClave

Esta función se ayuda de *endPasswordResetEmail(emailS)* el cual recibe un correo de entrada y enviará al email un correo donde podrá cambiar la clave (código 2.9).

```
public void cambiarClave(){
    authCambio.setLanguageCode("es");
    authCambio.sendPasswordResetEmail(emailS).addOnCompleteListener((task) → {
            if (task.isSuccessful()){

                Toast.makeText( context: RecuperarClaveActivity.this,  text: "Se envio un mensaje a su correo", Toast.LENGTH_SHORT).show();
                finish();
            }else {
                Toast.makeText( context: RecuperarClaveActivity.this,  text: "Este correo no existe", Toast.LENGTH_SHORT).show();
            }
            progreso.dismiss();
    });
```

Código 2.9. Función cambiarClave en RecuperarActivity

2.5.9. ACTIVIDAD PARA INICIAR SESIÓN (MAINACTIVITY)

Es la actividad para poder iniciar sesión e ir hacia el menú de cada usuario, dependiendo del rol se pedirá varios requisitos para poder ser completada.

A continuación, se muestra las líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, variables como “emailMA_S” que se utilizará para procesos de asignación de caracteres que varían, tres botones para llamar a las diferentes actividades y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.10.

```
public class MainActivity extends AppCompatActivity {

    private EditText emailMA,claveMA;
    private String emailMA_S,claveMA_S;
    private Button buttonIngresar;
    private FirebaseAuth authUser;
    private Button buttonRecuperar;
    private DatabaseReference dbUser;
```

Código 2.10. Código para MainActivity

- **Código para inicialización de elementos**

Se inicializa dos editText para clave y correo, dos botones para iniciar y recuperar, pero un botón directamente para registrar un nuevo usuario; autenticación y database de Firebase como se puede ver en el código 2.11.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    emailMA=(EditText)findViewById(R.id.editTextEmail);
    claveMA=(EditText)findViewById(R.id.editTextClave);
    buttonIngresar=(Button)findViewById(R.id.btnIniciar);
    buttonRecuperar=(Button)findViewById(R.id.btnRecuperar);
    dbUser=FirebaseDatabase.getInstance().getReference().child("USUARIOS");
    authUser=FirebaseAuth.getInstance();
```

Código 2.11. Inicialización de elementos en MainActivity

- **Código para botón Recuperar**

Al seleccionar este botón se abrirá la actividad para recuperar clave como se indica en el código 2.12.

```
buttonRecuperar.setOnClickListener((v) → {
        startActivity(new Intent( packageContext: MainActivity.this,RecuperarClaveActivity.class));
});
```

Código 2.12. Código para botón Recuperar en MainActivity

- **Código para botón Registrarse**

Al seleccionar este botón se abrirá la actividad para crear un usuario (código 2.13).

```
public void btnRegistrarse (View view){

    Intent siguiente=new Intent( packageContext: this,RegistroActivity.class);
    startActivity(siguiente);
}
```

Código 2.13. Código para botón Registrarse en MainActivity

- **Código para botón Ingresar**

Al seleccionar este botón se asegura de que las entradas de los campos correo y clave estén llenos y llamamos a la función *iniciarLogin()* como se observa en el código 2.14.

```
buttonIngresar.setOnClickListener((v) → {
        emailMA_S=emailMA.getText().toString();
        claveMA_S=claveMA.getText().toString();

        if(!emailMA_S.isEmpty()&&!claveMA_S.isEmpty()){
            //iremos a ActiviyTranscripcion
            iniciarLogin();
        }else {
            Toast.makeText( context: MainActivity.this,  text: "Complete Datos", Toast.LENGTH_SHORT).show();
        }

});
```

Código 2.14. Código para botón Ingresar en MainActivity

- **Código para botón Iniciar**

Esta función permitirá comprobar que la creación de usuario y confirmación por correo se haya cumplido gracias a *signInWithEmailAndPassword(emailMA_S,claveMA_S),* luego se hace una búsqueda en la base de datos para elegir el rol al que el usuario pertenece e iniciar la actividad correspondiente como se indica en el código 2.15.

```
public void iniciarLogin(){
    authUser.signInWithEmailAndPassword(emailMA_S,claveMA_S).addOnCompleteListener((task) → {
            if (task.isSuccessful()){
                FirebaseUser UserBD=authUser.getCurrentUser();
                if (UserBD.isEmailVerified()){
                    final String id=authUser.getCurrentUser().getUid();
                    dbUser.child("ADMINISTRADOR").child(id).addValueEventListener(new ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                            if(dataSnapshot.exists()){
                                dbUser.child("ADMINISTRADOR").child(id).child("Clave").
                                        setValue(claveMA_S);
                                startActivity(new Intent( packageContext: MainActivity.this,
                                        ActivityAdministrador.class));
                                finish();
                            }else {
                                dbUser.child("PRINCIPAL").child(id).addValueEventListener
                                        (new ValueEventListener() {
                                    @Override
                                    public void onDataChange(@NonNull DataSnapshot dataSnapshot2) {
                                        if (dataSnapshot2.exists()){
                                            dbUser.child("PRINCIPAL").child(id).child("Clave").
                                                    setValue(claveMA_S);
                                            startActivity(new Intent( packageContext: MainActivity.this,
                                                    ActivityTranscripcion.class));
                                            finish();
                                        }else{
                                            dbUser.child("FAMILIAR").child(id).addValueEventListener
                                                    (new ValueEventListener() {
                                                @Override
                                                public void onDataChange(@NonNull DataSnapshot dataSnapshot3) {
                                                    if (dataSnapshot3.exists()){
                                                        dbUser.child("FAMILIAR").child(id).child("Clave").setValue(claveMA_S);
                                                        startActivity(new Intent( packageContext: MainActivity.this,
                                                                ActivityFamiliar.class));
                                                        finish();
```

Código 2.15. Código para botón Iniciar en MainActivity

- **Código para onStart()**

Esta función ayudará a recuperar datos si algún usuario que ingresó con anterioridad, aunque se haya cerrado la aplicación. Su funcionamiento es verificar el usuario, hacer una búsqueda en la base de datos y enviar a la actividad correspondiente como se puede ver en el código 2.16.

```
protected void onStart() {
    super.onStart();
    if (authUser.getCurrentUser()!=null){
        FirebaseUser UserBD=authUser.getCurrentUser();
        if (UserBD.isEmailVerified()){
            final String id=authUser.getCurrentUser().getUid();
            dbUser.child("ADMINISTRADOR").child(id).addValueEventListener(
                    new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    if(dataSnapshot.exists()){
                        //dbUser.child("USUARIOS").child("ADMINISTRADOR").child(id).child("Clave").setValu
                        startActivity(new Intent( packageContext: MainActivity.this,
                                ActivityAdministrador.class));
                        finish();
                    }else {
                        dbUser.child("PRINCIPAL").child(id).addValueEventListener
                                (new ValueEventListener() {
                            @Override
                            public void onDataChange(@NonNull DataSnapshot dataSnapshot2) {
                                if (dataSnapshot2.exists()){
                                    //dbUser.child("USUARIOS").child("PRINCIPAL").child(id).child("Clave")
                                    startActivity(new Intent( packageContext: MainActivity.
                                            this,ActivityTranscripcion.class));
                                    finish();
                                }else{
                                    dbUser.child("FAMILIAR").child(id).addValueEventListener
                                            (new ValueEventListener() {
                                        @Override
                                        public void onDataChange(@NonNull DataSnapshot dataSnapshot3) {
                                            if (dataSnapshot3.exists()){
                                                // dbUser.child("USUARIOS").child("FAMILIAR").child(id).ch
                                                startActivity(new Intent( packageContext: MainActivity.this,
                                                        ActivityFamiliar.class));
                                                finish();
```

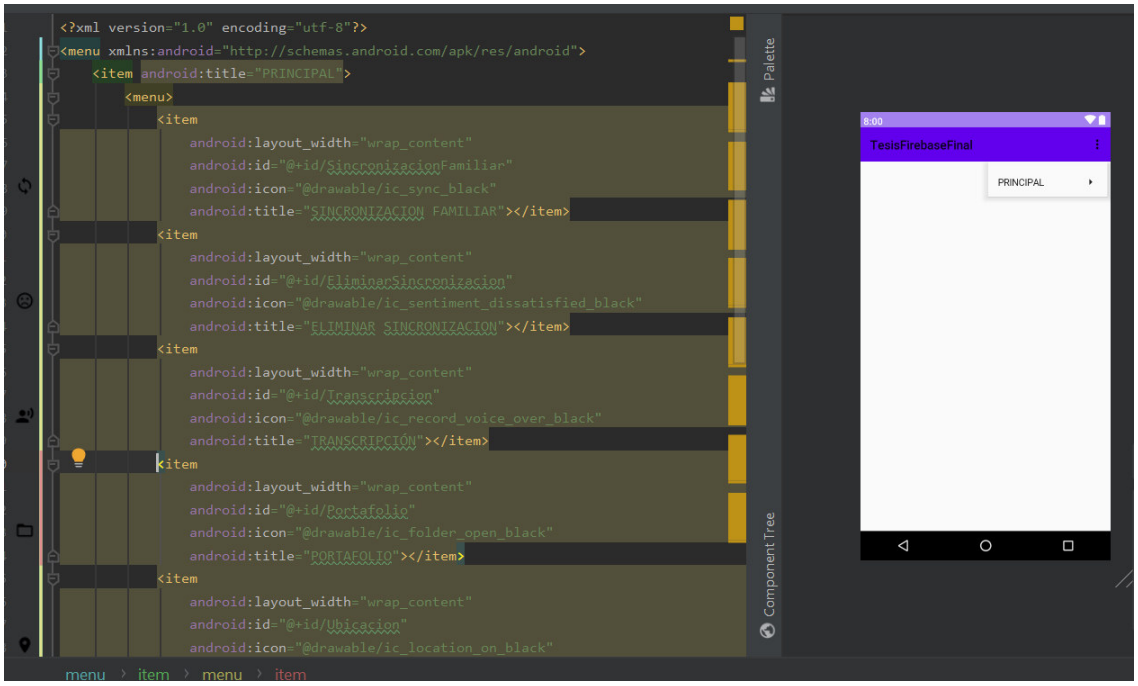
Código 2.16. OnStart en MainActivity

2.5.10. CÓDIGO DE COMPLEMENTOS PARA MENÚ DESLIZABLE

Esta sección es muy importante ya que se explicará la creación del menú deslizante en forma general. Esto se realiza para evitar redundancia ya que existe tres usuarios y se utiliza el mismo concepto para los tres.

- **Código de drawer_menu.xml**

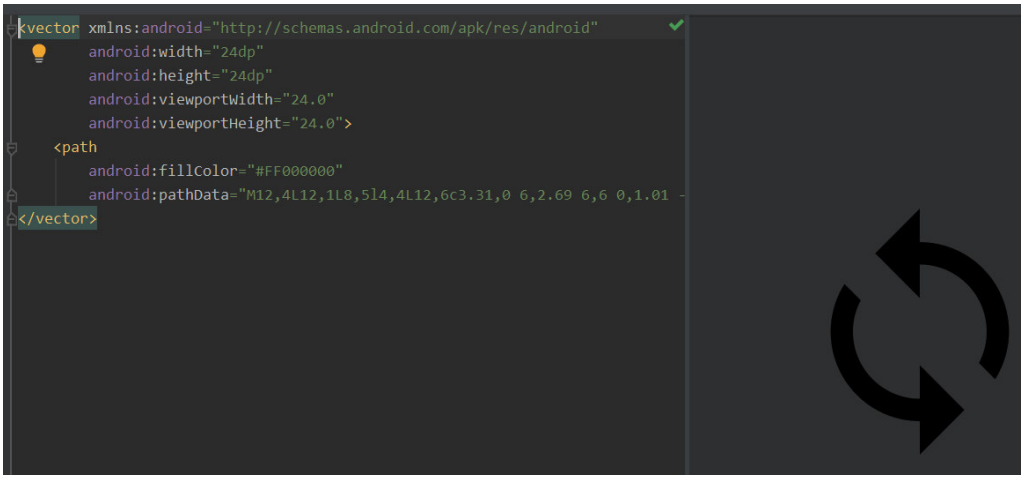
Es donde se agrega cada ítem del menú, es decir, si se necesita que el menú muestre “iniciar”, “jugar” y “salir” se creará 3 espacios. Se puede agregar un título y un icono, para el título se escribe *android:title="SINCRONIZACION FAMILIAR"* y para el icono *android:icon="@drawable/ic_sync_black"* (código 2.17).



Código 2.17. Drawer_menu.xml en menú

- **Código para icono**

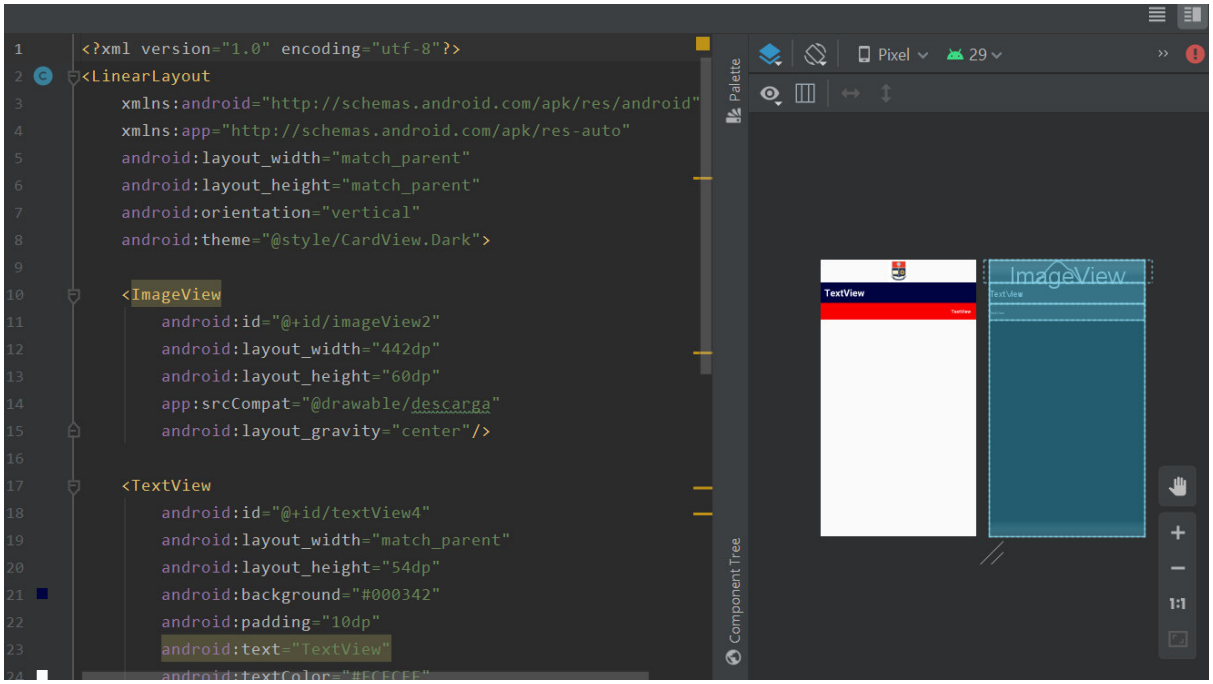
Se puede crear un nuevo icono con crear un *Vector Asset* (código 2.18).



Código 2.18. Nuevo icono para menú

- **Código de drawer_header.xml**

Esta sección sirve para agregar una cabecera en el menú, se puede crear a diseño del programador como se observa en el código 2.19.

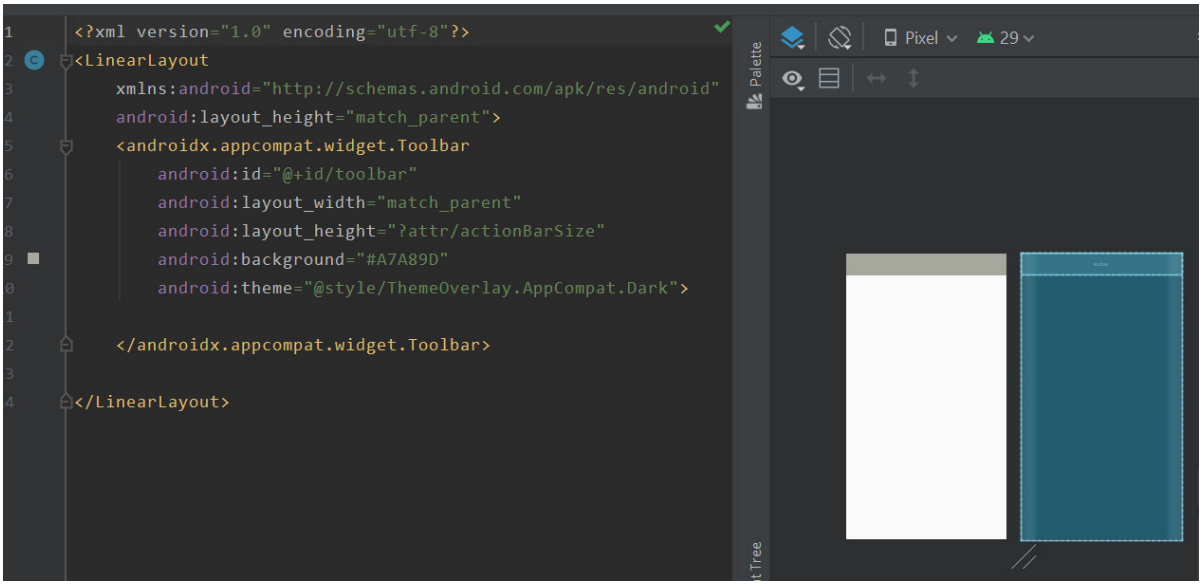



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:theme="@style/CardView.Dark">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="442dp"
        android:layout_height="60dp"
        app:srcCompat="@drawable/descarga"
        android:layout_gravity="center"/>

    <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="54dp"
        android:background="#000342"
        android:padding="10dp"
        android:text="TextView"
        android:textColor="#ECECEE"
```

Código 2.19. Drawer_header.xml en menú

- **Código de drawer_toolbar.xml**

Esto genera en la parte superior izquierda un botón que al pulsarlo abre el menú como se puede ver en el código 2.20.

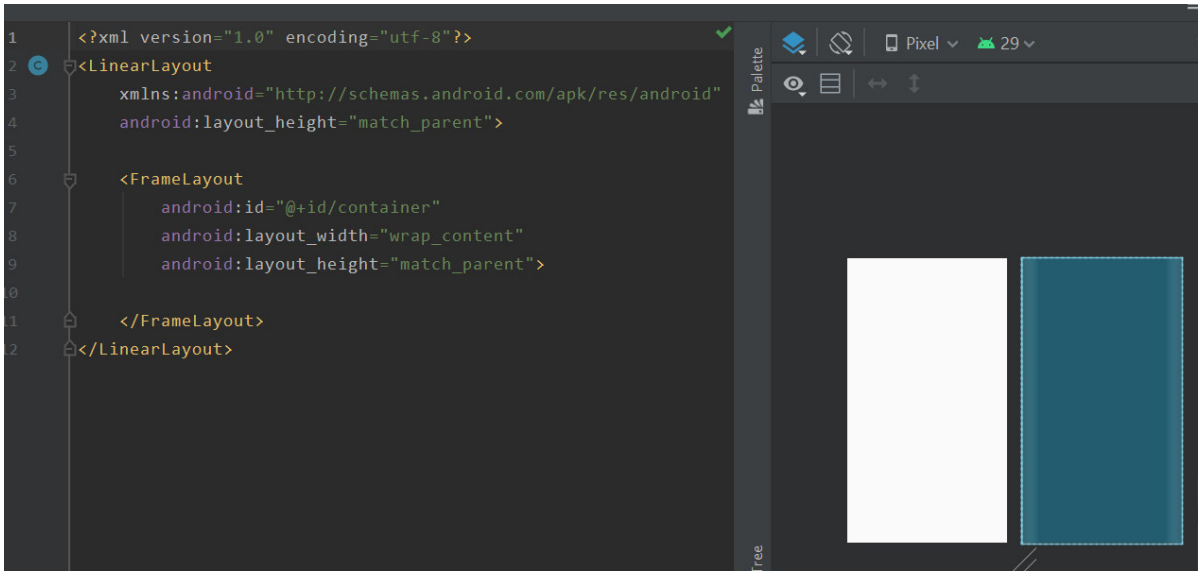



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent">
    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="#A7A89D"
        android:theme="@style/ThemeOverlay.AppCompat.Dark">

    </androidx.appcompat.widget.Toolbar>

</LinearLayout>
```

Código 2.20. Drawer_toolbar.xml en menú

- **Código de content_transcripcion.xml**

Este segmento genera un espacio donde se alojará toda la configuración del fragment que se envíe (código 2.21).



Código 2.21. Content_transcripcion.xml en menú

2.5.11. MENU ADMINISTRADOR (ACTIVITYADMINISTRADOR)

Es la actividad donde se programará el menú desplegable perteneciente al usuario Administrador y contendrá el llamado a cada fragment al seleccionar de cada ítem del menú.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, cargar un fragment inicial, cargar el menú de administrador completo y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.22.

```
public class ActivityAdministrador extends AppCompatActivity

    private DrawerLayout drawerLayout3;
    private ActionBarDrawerToggle actionBarDrawerToggle3;
    private Toolbar toolbar3;
    private NavigationView navigationView3;
    FragmentManager fragmentManager3;
    FragmentTransaction fragmentTransaction3;
    TextView txtViewHea3;
    //base de datos
    private DatabaseReference DbRef;
    private FirebaseAuth baseAutenticacion;
```

Código 2.22. Creación de atributos en ActivityAdministrador

- **Código para inicialización de elementos y cargar datos de usuario al menú**

En esta sección se carga los datos correspondientes al usuario haciendo un pedido a la base de datos con la línea de código 61, de la línea 78 a la 81 se carga el menú, la línea 83 permite

que los ítems del menú sean seleccionables y de la 85 a la 89 se carga el fragment inicial, en este caso es *AdminUsuariosFragment()* como se observa en el código 2.23.

```
57          baseAutenticacion= FirebaseAuth.getInstance();
58          final String id=baseAutenticacion.getCurrentUser().getUid();
59          DbRef= FirebaseDatabase.getInstance().getReference();
60
61          DbRef.child("USUARIOS").child("ADMINISTRADOR").child(id).addValueEventListener(new ValueEventListener() {
62              @Override
63              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
64                  if(dataSnapshot.exists()){
65                      View headerView = navigationView3.getHeaderView( index: 0);
66                      TextView navUsername = (TextView) headerView.findViewById(R.id.textView4);
67                      TextView correoFront=(TextView)headerView.findViewById(R.id.textView5);
68                      navUsername.setText(dataSnapshot.child("Apodo").getValue().toString());
69                      correoFront.setText(dataSnapshot.child("Correo").getValue().toString());
70                  }
71              }
72
73              @Override
74              public void onCancelled(@NonNull DatabaseError databaseError) {
75
76              }
77          });
78          actionBarDrawerToggle3=new ActionBarDrawerToggle( activity: this,drawerLayout3,toolbar3,"open3","close3");
79          drawerLayout3.addDrawerListener(actionBarDrawerToggle3);
80          actionBarDrawerToggle3.setDrawerIndicatorEnabled(true);
81          actionBarDrawerToggle3.syncState();
82          //establecer el evento onclick al navigationview
83          navigationView3.setNavigationItemSelectedListener(this);
84          //cargar fragment inicial
85          fragmentManager3=getSupportFragmentManager();
86          fragmentTransaction3=fragmentManager3.beginTransaction();
87          fragmentTransaction3.add(R.id.containerAdmin,new AdminUsuariosFragment());
88          fragmentTransaction3.commit();
89          txtViewHea3=(TextView) findViewById(R.id.textView4);
90      }
```

Código 2.23. Inicialización de elementos en ActivityAdministrador

- **Código para redirigir a ítem “Detalles”**

Gracias a *onNavigationItemSelected* es posible realizar este proceso, la línea 94 permitirá que el menú se guarde al momento de seleccionar un ítem, la línea 95 hace referencia al ítem seleccionado y verifica la selección; de la línea 96 a 100 se reemplaza el fragment con *AdminUsuariosFragment()* como se puede ver en el código 2.24.

```
92          @Override
93          public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
94              drawerLayout3.closeDrawer(GravityCompat.START);
95              if (menuItem.getItemId()==R.id.Detalles){
96                  DbRef=null;
97                  fragmentManager3=getSupportFragmentManager();
98                  fragmentTransaction3=fragmentManager3.beginTransaction();
99                  fragmentTransaction3.replace(R.id.containerAdmin,new AdminUsuariosFragment());
100                 fragmentTransaction3.commit();
101             }
```

Código 2.24. Redirigir a ítem “Detalles” en ActivityAdministrador

- **Código para redirigir a ítem “Reporte”, “Crear Administrador” y “Cerrar Sesión”**

Las líneas 103,110 y 114 hacen referencia al ítem seleccionado y verifica la selección; de la línea 105 a 108 se reemplaza el fragment con *ReporteGeneralFragment()*, la línea 111 llamará a la actividad *RegistroAdmin* y la línea 116 permite cerrar sesión (código 2.25).

```
103         if (menuItem.getItemId()==R.id.REPORTE){
104             DbRef=null;
105             fragmentManager3=getSupportFragmentManager();
106             fragmentTransaction3=fragmentManager3.beginTransaction();
107             fragmentTransaction3.replace(R.id.containerAdmin,new ReporteGeneralFragment());
108             fragmentTransaction3.commit();
109         }
110         if (menuItem.getItemId()==R.id.CrearAdministrador){
111             startActivity(new Intent( packageContext: ActivityAdministrador.this,RegistroAdmin.class));
112             //startActivity(new Intent(ActivityFamiliar.this,MapaActivity.class));
113         }
114         if (menuItem.getItemId()==R.id.CerrarSesion){
115             DbRef=null;
116             baseAutenticacion.signOut();
117 
118             startActivity(new Intent( packageContext: ActivityAdministrador.this,MainActivity.class));
119             finish();
120         }
121         return false;
122     }
123 }
```

Código 2.25. Redirigir a ítems en ActivityAdministrador

2.5.12. OBTENER LISTA DE NODOS DESDE DATABASE DE FIREBASE

Ya que en las siguientes secciones se hará el llamado a listas de diferentes puntos dentro de la base de datos y esto será redundante en varios casos, se creó esta parte donde se aclararán temas como un “singlerow”,” modelo” y “adaptador”; se explica a continuación:

- **Singlerow**

Es una sección del proyecto de archivo *.xml* que permite visualizar la información de un sub nodo de la base de datos. Para ejemplo se tiene un nodo principal llamado ADMINISTRADOR y dos sub nodos como se observa en la Figura 2.62., pues permitirá visualizar los datos que se envíe del sub nodo hacia singlerow y así se puede llamar a cada sub nodo y reemplazar los datos del singlerow cuando se desee.

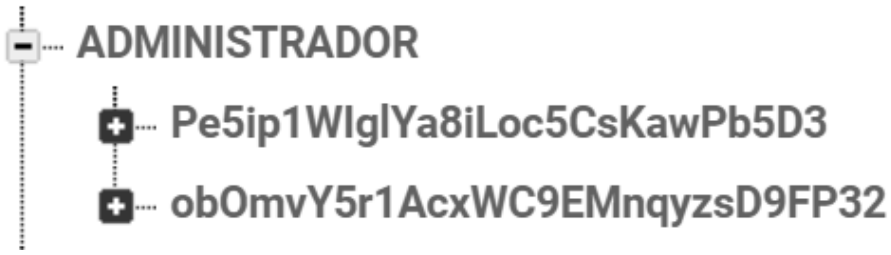
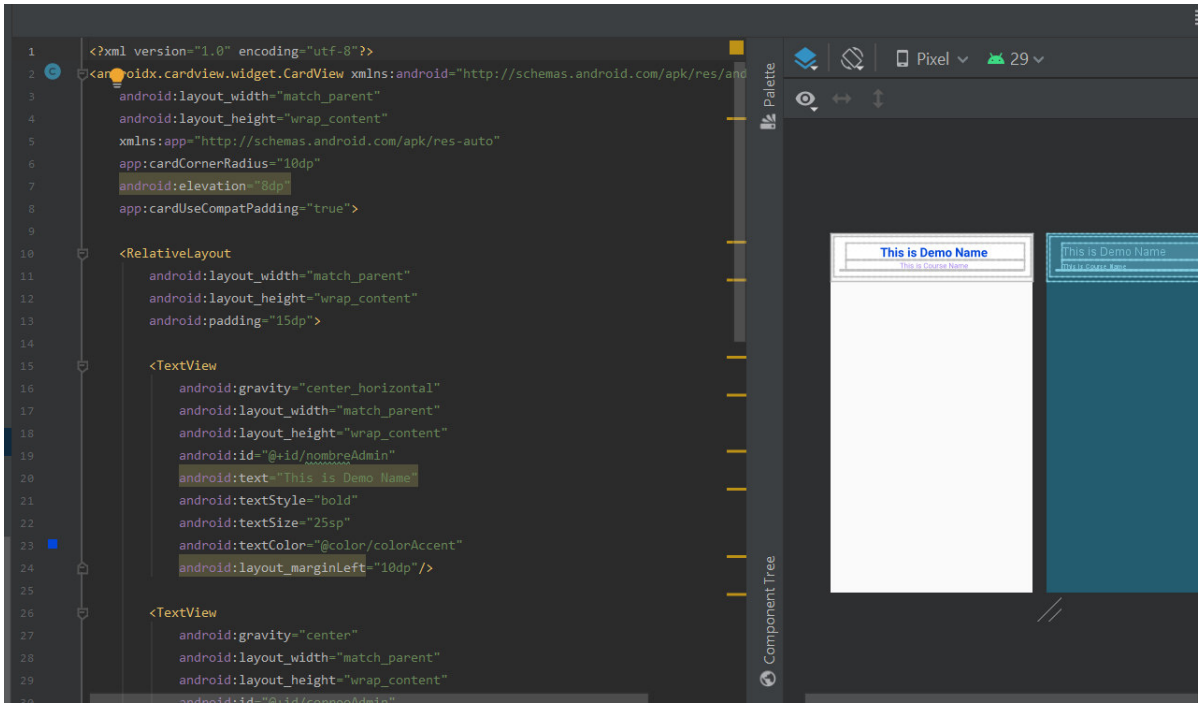


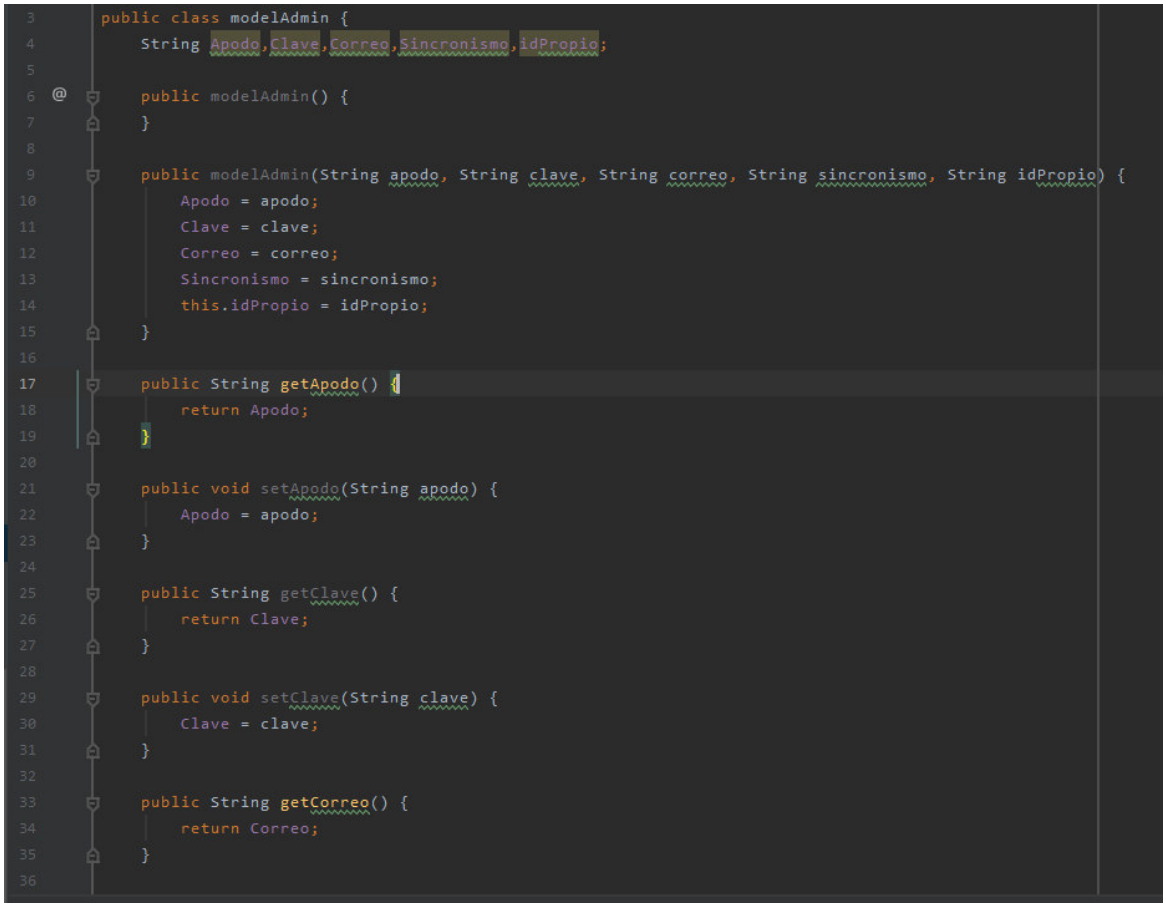
Figura 2.62. Nodo y subnodos



Código 2.26. Singlerowadmin.xml en el proyecto

- **Modelo**

Es una clase que permite poder obtener los diferentes atributos de un nodo o un subnodo, hay que aclarar que se utilizará varios modelos dependiendo que datos necesitemos, pero tendrán el mismo formato. Una parte importante es que los atributos de este modelo deben tener el mismo nombre del atributo de la base de datos ya que si no se tendrán errores al momento de compilar el proyecto. Para ello se indica un ejemplo de codificación de un modelo en la siguiente figura donde se creó un constructor con todos los atributos, get y set para cada atributo como se observa en el código 2.27.

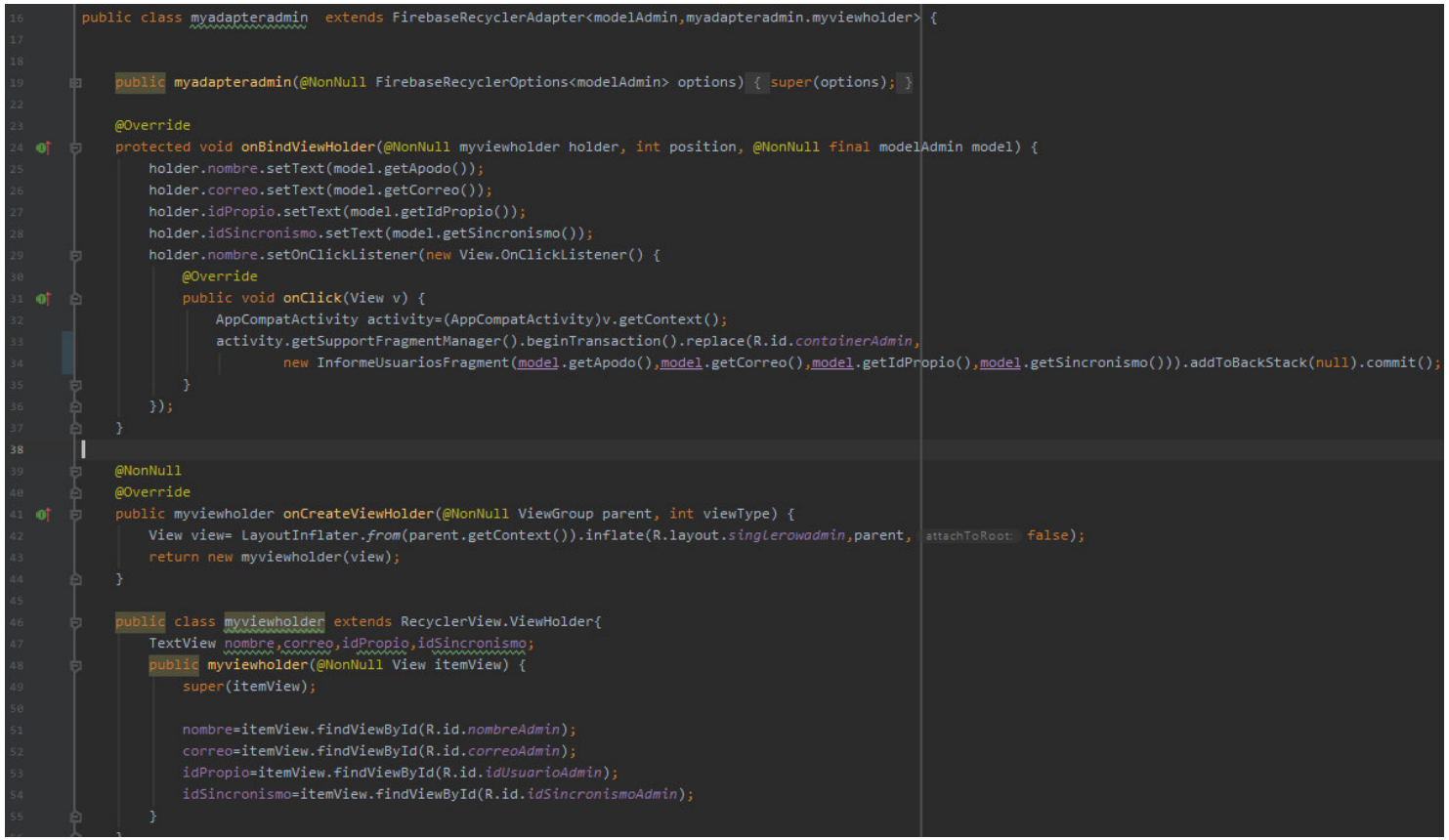


```
3   public class modelAdmin {
4       String Apodo,Clave,Correo,Sincronismo,idPropio;
5
6   @   public modelAdmin() {
7       }
8
9       public modelAdmin(String apodo, String clave, String correo, String sincronismo, String idPropio) {
10          Apodo = apodo;
11          Clave = clave;
12          Correo = correo;
13          Sincronismo = sincronismo;
14          this.idPropio = idPropio;
15      }
16
17      public String getApodo() {
18          return Apodo;
19      }
20
21      public void setApodo(String apodo) {
22          Apodo = apodo;
23      }
24
25      public String getClave() {
26          return Clave;
27      }
28
29      public void setClave(String clave) {
30          Clave = clave;
31      }
32
33      public String getCorreo() {
34          return Correo;
35      }
36
```

Código 2.27. ModelAdmin en el proyecto

- **Adaptador**

Para entender esta sección se observará el código 2.28. Esta clase *adaptador* extiende de *FirebaseRecyclerAdapter* la cual permite obtener datos de Firebase referentes a un *modelo* y aun *singlerow;* para ello de la línea 45 a la 53 lo que se codifica es crear los atributos que se tiene en el *singlerow* en este caso son *nombre, correo, idPropio, idSincronismo* que en *singlerow* son textView y estos atributos se los inicializa o enlaza con los atributos propios de *singlerow.* De la línea 24 a la 28 se obtien los datos de los atributos utilizando los get & set del *modelo* y guardándolos dentro de un objeto que se llamará *holder.* De la línea 40 a 42 se cargará estos datos del objeto *holder* en el *singlerow* y se enviará a modo de vista. De la línea 29 a la 34 permitirá que cuando se presione el elemento de toda la lista nos redireccione a otro evento, este puede ser una actividad o un fragment; en la el código 2.28 se dirige a *InformeUsuariosFragmen* enviando los cuatro atributos para que este nuevo fragment los pueda utilizar de la manera que desee.



```
public class myadapteradmin  extends FirebaseRecyclerAdapter<modelAdmin,myadapteradmin.myviewholder> {


    public myadapteradmin(@NonNull FirebaseRecyclerOptions<modelAdmin> options) { super(options); }

    @Override
    protected void onBindViewHolder(@NonNull myviewholder holder, int position, @NonNull final modelAdmin model) {
        holder.nombre.setText(model.getApodo());
        holder.correo.setText(model.getCorreo());
        holder.idPropio.setText(model.getIdPropio());
        holder.idSincronismo.setText(model.getSincronismo());
        holder.nombre.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                AppCompatActivity activity=(AppCompatActivity)v.getContext();
                activity.getSupportFragmentManager().beginTransaction().replace(R.id.containerAdmin,
                        new InformeUsuariosFragment(model.getApodo(),model.getCorreo(),model.getIdPropio(),model.getSincronismo())).addToBackStack(null).commit();
            }
        });
    }

    @NonNull
    @Override
    public myviewholder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view= LayoutInflater.from(parent.getContext()).inflate(R.layout.singlerowadmin,parent, attachToRoot: false);
        return new myviewholder(view);
    }

    public class myviewholder extends RecyclerView.ViewHolder{
        TextView nombre,correo,idPropio,idSincronismo;
        public myviewholder(@NonNull View itemView) {
            super(itemView);

            nombre=itemView.findViewById(R.id.nombreAdmin);
            correo=itemView.findViewById(R.id.correoAdmin);
            idPropio=itemView.findViewById(R.id.idUsuarioAdmin);
            idSincronismo=itemView.findViewById(R.id.idSincronismoAdmin);
        }
    }
```

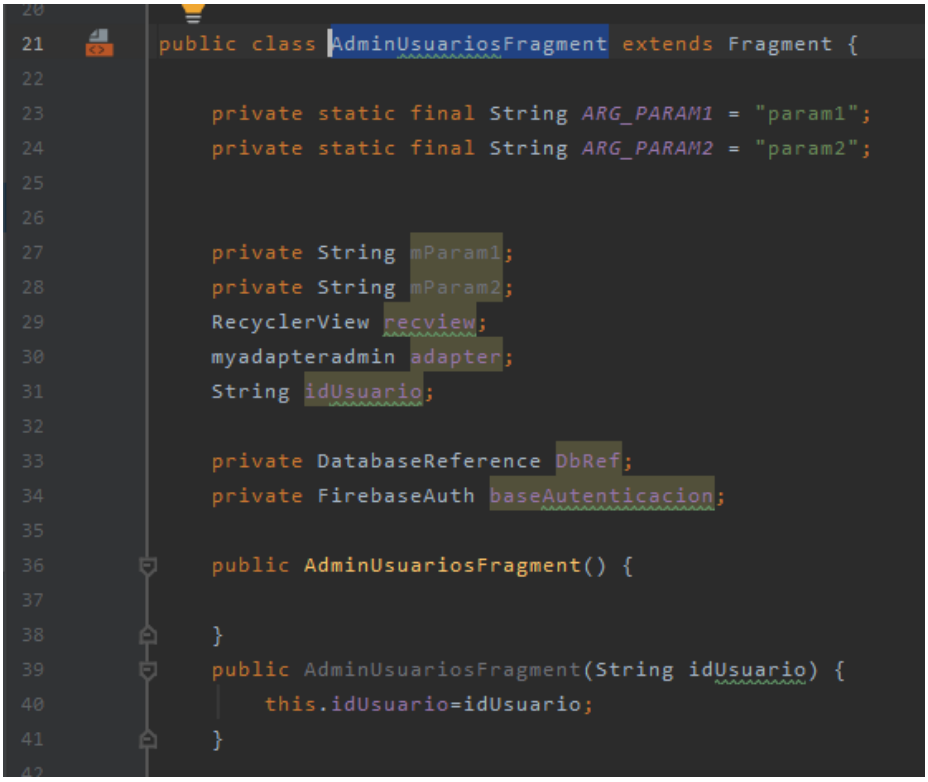
Código 2.28. AdapterAdmin en el proyecto

2.5.13. LISTA DE USUARIOS PRINCIPALES (ADMINUSUARIOSFRAGMENT)

En este fragment se obtendrá una lista de todos los usuarios Principales que existan en la base de datos, además se podrá seleccionar un usuario y ver detalles generales.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para cargar la lista completa de los usuarios, autenticación y base de datos de Firebase. En la línea 39 se ha creado un constructor del fragment ya que al utilizar Firebase se necesita enviar y recibir atributos que identifiquen un nodo en la base de datos como se indica en el código 2.29.

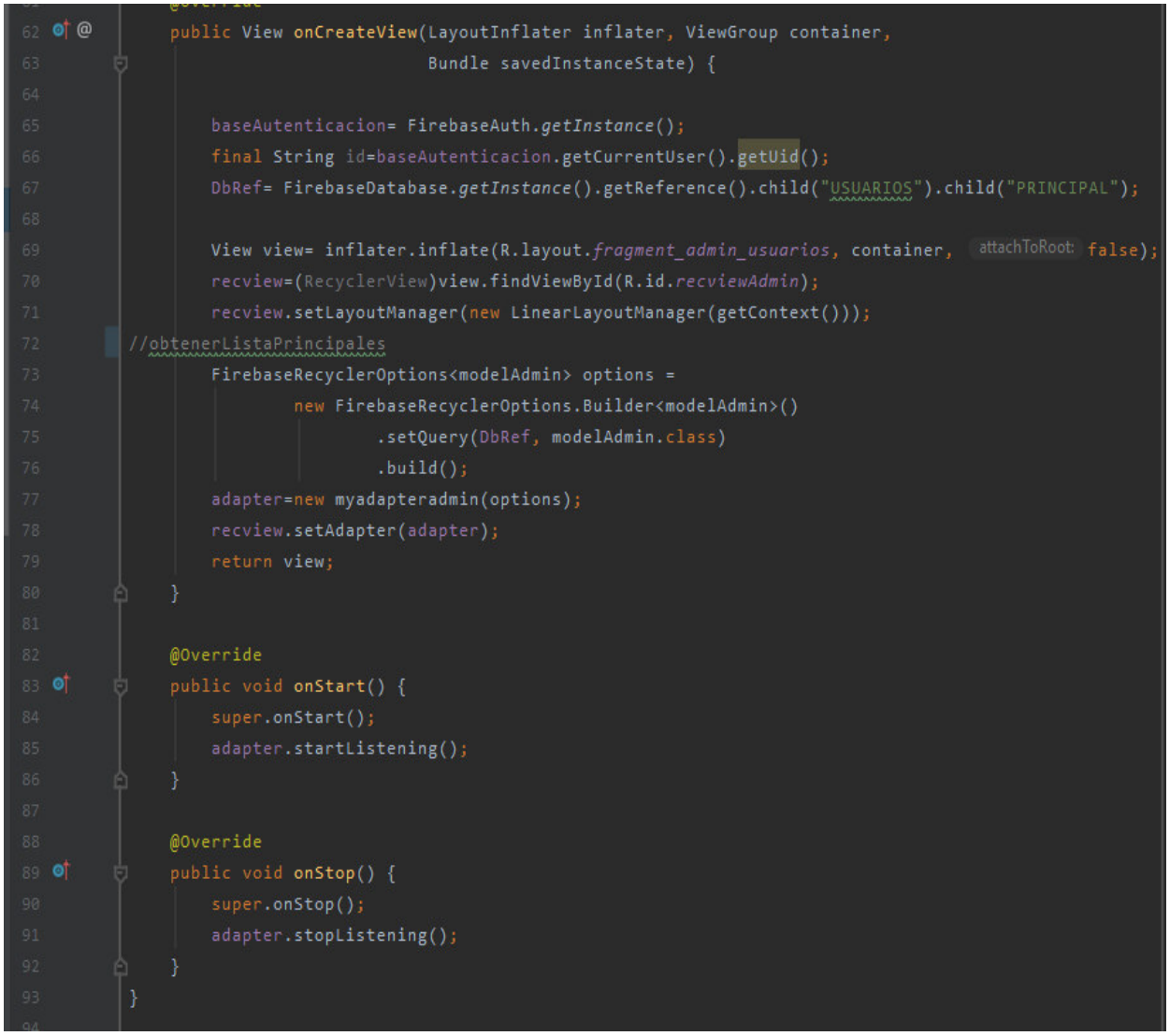


```
21  public class AdminUsuariosFragment extends Fragment {
22
23      private static final String ARG_PARAM1 = "param1";
24      private static final String ARG_PARAM2 = "param2";
25
26
27      private String mParam1;
28      private String mParam2;
29      RecyclerView recview;
30      myadapteradmin adapter;
31      String idUsuario;
32
33      private DatabaseReference DbRef;
34      private FirebaseAuth baseAutenticacion;
35
36      public AdminUsuariosFragment() {
37
38      }
39      public AdminUsuariosFragment(String idUsuario) {
40          this.idUsuario=idUsuario;
41      }
```

Código 2.29. Creación de atributos en AdminUsuariosFragment

- **Código para funcionamiento de AdminUsuariosFragment**

Para mejor entendimiento de esta sección se puede decir que de la línea 65 a 67 se apunta a la base de datos en específico al nodo de usuarios Principales, de la línea 69 a 71 se carga la parte visual del fragment, de la línea 73 a 79 y gracias a *FirebaseRecyclerOptions* que permite pasar un modelo apuntando hacia un nodo se puede obtener todos los subnodos de este nodo individualmente para luego mostrar con el adaptador como se puede ver en el código 2.30.



```
62  @   public View onCreateView(LayoutInflater inflater, ViewGroup container,
63                               Bundle savedInstanceState) {
64
65          baseAutenticacion= FirebaseAuth.getInstance();
66          final String id=baseAutenticacion.getCurrentUser().getUid();
67          DbRef= FirebaseDatabase.getInstance().getReference().child("USUARIOS").child("PRINCIPAL");
68
69          View view= inflater.inflate(R.layout.fragment_admin_usuarios, container,  attachToRoot: false);
70          recview=(RecyclerView)view.findViewById(R.id.recviewAdmin);
71          recview.setLayoutManager(new LinearLayoutManager(getContext()));
72  //obtenerListaPrincipales
73          FirebaseRecyclerOptions<modelAdmin> options =
74                  new FirebaseRecyclerOptions.Builder<modelAdmin>()
75                          .setQuery(DbRef, modelAdmin.class)
76                          .build();
77          adapter=new myadapteradmin(options);
78          recview.setAdapter(adapter);
79          return view;
80      }
81
82      @Override
83      public void onStart() {
84          super.onStart();
85          adapter.startListening();
86      }
87
88      @Override
89      public void onStop() {
90          super.onStop();
91          adapter.stopListening();
92      }
93  }
```

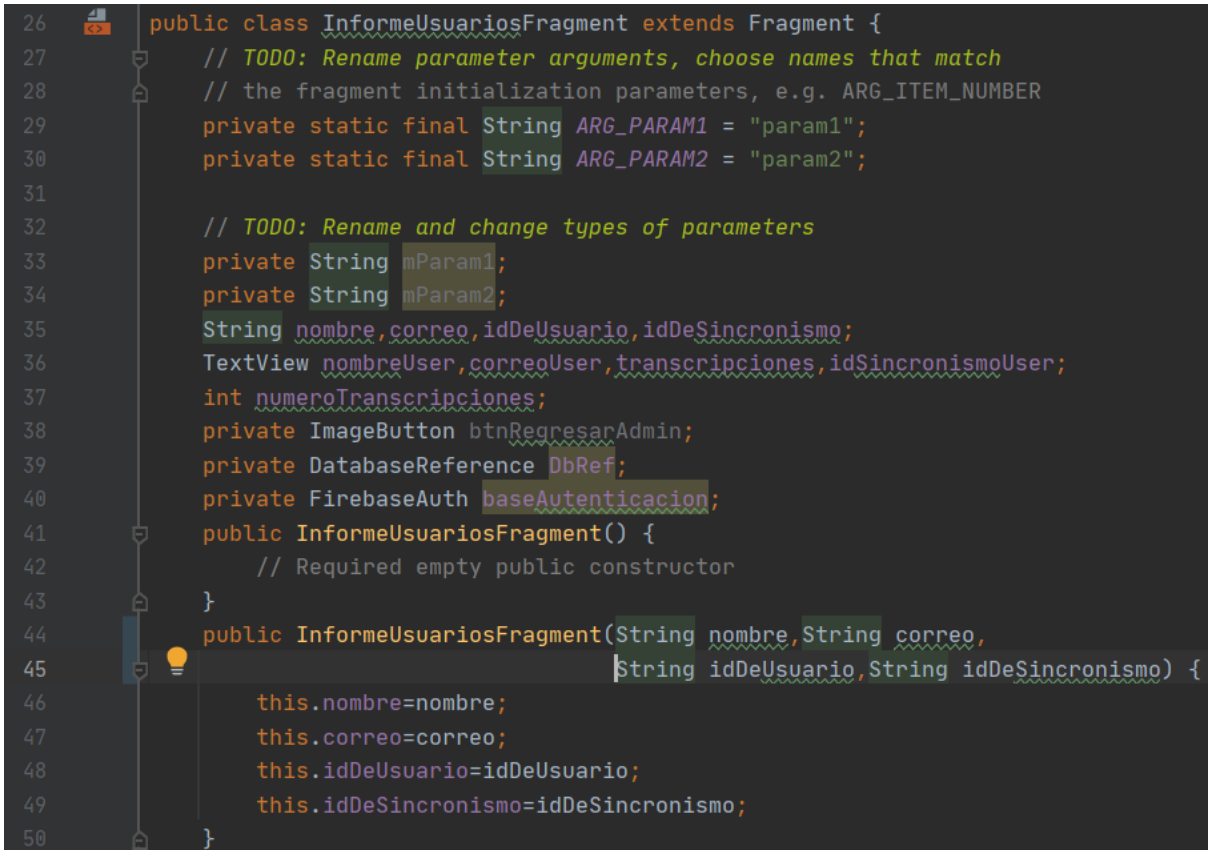
Código 2.30. Funcionamiento de AdminUsuariosFragment

2.5.14. INFORMACIÓN (INFORMEUSUARIOSFRAGMENT)

Este Fragment permitirá obtener información del usuario Principal que se seleccione en el AdminUsuariosFragment y mostrarla en pantalla de una manera ordenada y simplificada.

- **Código para creación de atributos**

Se crea atributos para inicializar los diferentes objetos, atributos para autenticación y base de datos referentes a Firebase y en la línea 44 se crea un constructor que recibe cuatro atributos; se genera esto ya que el *Adaptador* correspondiente llamará a este fragment enviándole los cuatro atributos que se obtuvo desde firebase (código 2.31).

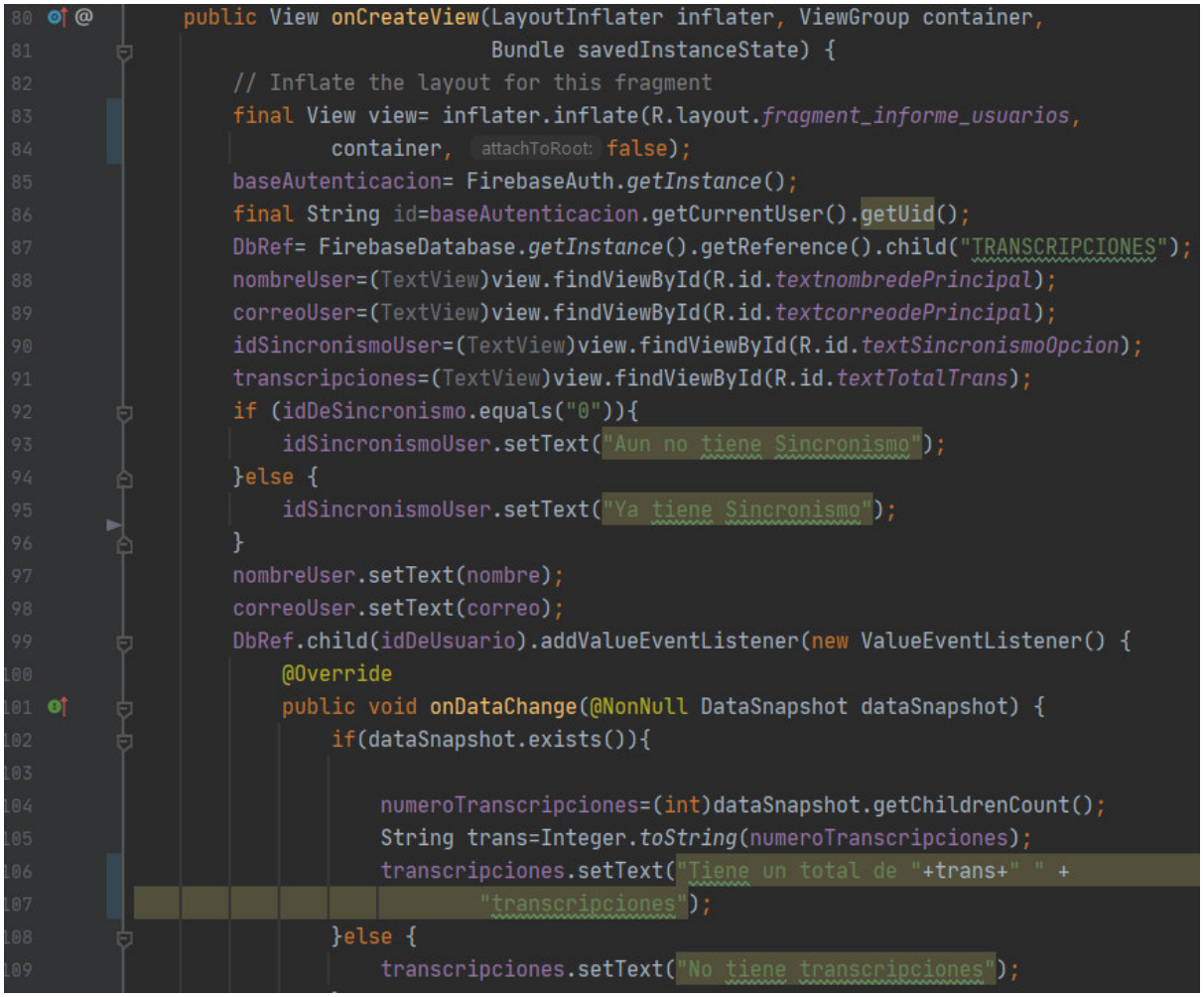


```
26  public class InformeUsuariosFragment extends Fragment {
27      // TODO: Rename parameter arguments, choose names that match
28      // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
29      private static final String ARG_PARAM1 = "param1";
30      private static final String ARG_PARAM2 = "param2";
31
32      // TODO: Rename and change types of parameters
33      private String mParam1;
34      private String mParam2;
35      String nombre,correo,idDeUsuario,idDeSincronismo;
36      TextView nombreUser,correoUser,transcripciones,idSincronismoUser;
37      int numeroTranscripciones;
38      private ImageButton btnRegresarAdmin;
39      private DatabaseReference DbRef;
40      private FirebaseAuth baseAutenticacion;
41      public InformeUsuariosFragment() {
42          // Required empty public constructor
43      }
44      public InformeUsuariosFragment(String nombre,String correo,
45                                     String idDeUsuario,String idDeSincronismo) {
46          this.nombre=nombre;
47          this.correo=correo;
48          this.idDeUsuario=idDeUsuario;
49          this.idDeSincronismo=idDeSincronismo;
50      }
```

Código 2.31. Creación de atributos en InformeUsuariosFragment

- **Código para funcionamiento**

De la línea 85 a la 92 se inicializa todos los atributos, de la línea 101 a 114 se llama a la base de datos para hacer un conteo de transcripciones pertenecientes al usuario seleccionado, se cambia las variables y se muestra en pantalla como se observa en el código 2.32.



```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    final View view= inflater.inflate(R.layout.fragment_informe_usuarios,
            container,  attachToRoot: false);
    baseAutenticacion= FirebaseAuth.getInstance();
    final String id=baseAutenticacion.getCurrentUser().getUid();
    DbRef= FirebaseDatabase.getInstance().getReference().child("TRANSCRIPCIONES");
    nombreUser=(TextView)view.findViewById(R.id.textnombredePrincipal);
    correoUser=(TextView)view.findViewById(R.id.textcorreodePrincipal);
    idSincronismoUser=(TextView)view.findViewById(R.id.textSincronismoOpcion);
    transcripciones=(TextView)view.findViewById(R.id.textTotalTrans);
    if (idDeSincronismo.equals("0")){
        idSincronismoUser.setText("Aun no tiene Sincronismo");
    }else {
        idSincronismoUser.setText("Ya tiene Sincronismo");
    }
    nombreUser.setText(nombre);
    correoUser.setText(correo);
    DbRef.child(idDeUsuario).addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists()){

                numeroTranscripciones=(int)dataSnapshot.getChildrenCount();
                String trans=Integer.toString(numeroTranscripciones);
                transcripciones.setText("Tiene un total de "+trans+" " +
                        "transcripciones");
            }else {
                transcripciones.setText("No tiene transcripciones");
```

Código 2.32. Funcionamiento de InformeUsuariosFragment

2.5.15. REPORTE GENERAL (REPORTEGENERALFRAGMENT)

En este Fragment se captura datos pertenecientes a toda la base de datos y se los muestra por pantalla de manera ordenada.

- **Código para creación de atributos**

Se crea atributos para inicialización, autenticación y base de datos, estos diferentes atributos indicarán cinco aspectos fundamentales respecto a la base de datos los cuales son: total de usuarios de los tres roles, total de transcripciones y total de mensajes como se indica en el código 2.33.

```
27  public class ReporteGeneralFragment extends Fragment {
28      // TODO: Rename parameter arguments, choose names that match
29      // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
30      private static final String ARG_PARAM1 = "param1";
31      private static final String ARG_PARAM2 = "param2";
32
33      // TODO: Rename and change types of parameters
34      private String mParam1;
35      private String mParam2;
36      String sPrincipal,sFamiliar,sAdmin,sTranscripcion,sMensajes;
37      TextView tAdmin,tFamiliar,tPrincipal,tTranscripciones,tMensajes;
38      int numAdmin,numFamiliar,numPrincipal,numTranscripciones,contadorTranscripciones=0,numMensajes,contadorMensajes=0;
39      private ImageButton btnRegresarAdmin;
40      private DatabaseReference DbRef;
41      private FirebaseAuth baseAutenticacion;
42
```

Código 2.33. Código para Reporte GeneralFragment

- **Código para funcionamiento**

De la línea 78 a 86 se inicializa los objetos entre ellos el total de usuarios Administradores, Principales, Familiares; el total de mensajes almacenados en la base de datos y el total de transcripciones en la base de datos. En la línea 88 se hace un llamado a *addListenerForSingleValueEvent* para obtener datos de la base de datos; de la línea 94 a la 104 se obtiene el total de los tres tipos de usuarios, en la línea 105 y 111 se crea dos *for* internos para hacer un conteo y encontrar el número total de transcripciones y el total de mensajes respectivamente como se puede ver en el código 2.34.

```
        // Inflate the layout for this fragment
        final View view= inflater.inflate(R.layout.fragment_reporte_general, container,  attachToRoot: false);
        baseAutenticacion= FirebaseAuth.getInstance();
        final String id=baseAutenticacion.getCurrentUser().getUid();
        DbRef= FirebaseDatabase.getInstance().getReference();
        tAdmin=(TextView)view.findViewById(R.id.totalAdmin);
        tFamiliar=(TextView)view.findViewById(R.id.totalFamiliar);
        tPrincipal=(TextView)view.findViewById(R.id.totalPrincipal);
        tTranscripciones=(TextView)view.findViewById(R.id.totalTranscripciones);
        tMensajes=(TextView)view.findViewById(R.id.totalMensajes);

        DbRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                contadorTranscripciones=0;
                if(dataSnapshot.exists()){

                    numAdmin=(int)dataSnapshot.child("USUARIOS").child("ADMINISTRADOR").getChildrenCount();
                    sAdmin=Integer.toString(numAdmin);
                    numPrincipal=(int)dataSnapshot.child("USUARIOS").child("PRINCIPAL").getChildrenCount();
                    sPrincipal=Integer.toString(numPrincipal);
                    numFamiliar=(int)dataSnapshot.child("USUARIOS").child("FAMILIAR").getChildrenCount();
                    sFamiliar=Integer.toString(numFamiliar);
                    tAdmin.setText("Usuarios Administradores: "+sAdmin);
                    tPrincipal.setText("Usuarios Principales: "+sPrincipal);
                    tFamiliar.setText("Usuarios Familiares: "+sFamiliar);
                    contadorTranscripciones=0;
                    contadorMensajes=0;
                    for (DataSnapshot snapshot:dataSnapshot.child("TRANSCRIPCIONES").getChildren()){
                    numTranscripciones=(int)snapshot.getChildrenCount();
                    contadorTranscripciones=contadorTranscripciones+numTranscripciones;
                    }
                    sTranscripcion=Integer.toString(contadorTranscripciones);
                    tTranscripciones.setText("Total de Transcripciones: "+sTranscripcion);
                    for (DataSnapshot snapshot2:dataSnapshot.child("MENSAJES").getChildren()){
                        numMensajes=(int)snapshot2.getChildrenCount();
                        contadorMensajes=contadorMensajes+numMensajes;
                    }
                    sMensajes=Integer.toString(contadorMensajes);
                    tMensajes.setText("Total de Mensajes: "+sMensajes);
```

Código 2.34. Funcionamiento de ReporteGeneralFragment

2.5.16. REGISTRO DE USUARIOS ADMINISTRADORES (REGISTROADMIN)

Es la actividad para el registro de usuarios administradores dentro de la aplicación, se pedirá varios requisitos para poder ser completada.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, variables como “claveS” que se utilizará para procesos de asignación de caracteres que varían y atributos pertenecientes a la base de datos (código 2.35).

```
public class RegistroAdmin extends AppCompatActivity {
    private Button buttonContinuarAdmin;
    private EditText clave,claveRep,apodo,email;
    private FirebaseAuth baseAutenticacion;
    String claveS;
    String claveRepS;
    String emailS;
    String sincronismo="0";
    String apodoS,claseUsuario;
    private DatabaseReference baseDatos;
```

Código 2.35. Creación de variables RegistroActivity

- **Código para inicialización de elementos**

Se inicializa cuatro editText para nombre, email, clave y confirmación de clave; un botón para continuar con el registro del usuario y los dos campos para autenticación y base de datos de Firebase como se observa en el código 2.36.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_registro_admin);
    clave=(EditText) findViewById(R.id.editTextClaveAdmin);
    claveRep=(EditText) findViewById(R.id.editTextClaveRepAdmin);
    apodo=(EditText)findViewById(R.id.editTextApodoAdmin);
    email=(EditText)findViewById(R.id.editTextEmailAdmin);
    buttonContinuarAdmin=(Button) findViewById(R.id.btnContinuarAdmin);
    baseAutenticacion= FirebaseAuth.getInstance();
    baseDatos= FirebaseDatabase.getInstance().getReference();
```

Código 2.36. Inicialización de elementos en RegistroActivity

- **Código botón Continuar Admin**

Al seleccionar el botón Continuar Admin se captura los datos ingresados y se los almacena en variables de tipo string, se comprueba que los campos no están vacíos y luego que las contraseñas coincidan y tengan un cierto número de caracteres como se indica en el código 2.37.

```
56      public void btnContinuarAdmin (){
57          claveS=clave.getText().toString();
58          claveRepS=claveRep.getText().toString();
59          emailS=email.getText().toString();
60          apodoS=apodo.getText().toString();
61          if(!claveRepS.isEmpty()&&!claveS.isEmpty()&&!apodoS.isEmpty()&&!emailS.isEmpty()){
62              if (claveS.equals(claveRepS)){
63                  if (claveS.length()>5){
64
```

Código 2.37. Botón Continuar Admin en RegistroActivity

- **Alerta de diálogo botón Continuar**

Se crea un diálogo de alerta para confirmar que el correo ingresado anteriormente es el correcto, al seleccionar *SI* pasamos a la función *registrarUsuarioAdmin()* y al seleccionar *NO* se mantiene en el Activity para hacer un cambio a cualquier ítem como se puede ver en el código 2.38.

```
66      final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder( context: RegistroAdmin.this);
67      confirmacionDialog.setMessage(Html.fromHtml( source: "Confirma que el correo es: " + "<b>" + emailS + "</b>")).setCancelable(false)
68              .setPositiveButton( text: "Si", (dialog, which) -> {
71                      registrarUsuarioAdmin();
72              }).setNegativeButton( text: "No", (dialog, which) -> {
76              dialog.cancel();
77      });
79      AlertDialog alerta = confirmacionDialog.create();
80      alerta.setTitle("Cofirmación");
81      alerta.show();
```

Código 2.38. Alerta de diálogo en RegistroActivity

- **Código para función registrarUsuarioAdmin**

En esta función se llama a *createUserWithEmailAndPassword(emailS,claveS)* de autenticación en Firebase la cual pide un email y una clave para crear el usuario y registrarlo directamente en el módulo de autenticación en Firebase, además, se apunta a Database Realtime de Firebase, se crea un mapa de atributos *Map<String,Object>* el cual acepta 5 objetos para más tarde guardar en el nodo perteneciente al usuario dentro de la base de datos; también se envía un correo de confirmación de cuenta al email perteneciente al usuario creado. De la línea 139 a 144 se finaliza la actividad borrando cualquier dato guardado en cache al presionar el botón *atrás* en cualquier dispositivo como se observa en el código 2.39 y 2.40.

```
public void registrarUsuarioAdmin(){

    baseAutenticacion.createUserWithEmailAndPassword(email1S,claveS).addOnCompleteListener((task) → {
            if (task.isSuccessful()){
                //baseAutenticacion.setLanguageCode("es");
                String id=baseAutenticacion.getCurrentUser().getUid();
                Map<String,Object> mapUsuario=new HashMap<>();
                mapUsuario.put( k: "idPropio",id);
                mapUsuario.put( k: "Apodo",apodoS);
                mapUsuario.put( k: "Correo",email1S);
                mapUsuario.put( k: "Clave",claveS);
                mapUsuario.put( k: "Sincronismo",sincronismo);

                baseDatos.child("USUARIOS").child("ADMINISTRADOR").child(id).setValue(mapUsuario).addOnCompleteListener((task2) → {
                        if (task2.isSuccessful()){
                            //ir a activity que ya transcribe
                            FirebaseUser UserBD=baseAutenticacion.getCurrentUser();
                            UserBD.sendEmailVerification();
                            Toast.makeText( context: RegistroAdmin.this,  text: "Usuario creado, confirme en su correo por favor", Toast.LENGTH_SHORT).show();
                            baseDatos=null;
                            finish();
                        }else{
                            Toast.makeText( context: RegistroAdmin.this,  text: "Por el momento no es posible crear usuarios, disculpas", Toast.LENGTH_SHORT).show();
                        }
```

Código 2.39. Creación de variables RegistroActivity

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if(keyCode == event.KEYCODE_BACK){
        baseDatos=null;
        finish();
    }
    return super.onKeyDown(keyCode, event);
}
```

Código 2.40. onKeyDown en RegistroActivity

2.5.17. MENÚ FAMILIAR (ACTIVITYFAMILIAR)

Es la actividad donde se programará el menú desplegable perteneciente al usuario Familiar y contendrá el llamado a cada fragment al seleccionar cada ítem del menú.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, cargar el menú familiar completo y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.41.

```
39  public class ActivityFamiliar extends AppCompatActivity implements NavigationView.OnNavigationItemSelectedListener{
40      private DrawerLayout drawerLayout2;
41      private ActionBarDrawerToggle actionBarDrawerToggle2;
42      private Toolbar toolbar2;
43      private NavigationView navigationView2;
44      FragmentManager fragmentManager2;
45      FragmentTransaction fragmentTransaction2;
46      TextView txtViewHea2;
47      //base de datos
48      private DatabaseReference DbRef;
49      private FirebaseAuth baseAutenticacion;
50      String condicionSincronismoPrincipal;
```

Código 2.41. Creación de atributos en ActivityFamiliar

- **Código para inicialización de elementos y cargar datos de usuario al menú**

En esta sección se carga los datos correspondientes al usuario haciendo un pedido a la base de datos con la línea de código 66, de la línea 87 a la 90 se carga el menú, la línea 92 permite que los ítems del menú sean seleccionables y de la 94 a la 97 se carga el fragment inicial, en este caso es *FamiliarFragment()* el cual está vacío como se puede ver en el código 2.42.

```
58          baseAutenticacion= FirebaseAuth.getInstance();
59          final String id=baseAutenticacion.getCurrentUser().getUid();
60          DbRef= FirebaseDatabase.getInstance().getReference( path: "USUARIOS").child("FAMILIAR");
61
62          DbRef.child(id).addValueEventListener(new ValueEventListener() {
63              @Override
64              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
65                  if(dataSnapshot.exists()){
66                      View headerView = navigationView2.getHeaderView( index: 0);
67                      TextView navUsername = (TextView) headerView.findViewById(R.id.textView4);
68                      TextView correoFront=(TextView)headerView.findViewById(R.id.textView5);
69                      condicionSincronismoPrincipal=dataSnapshot.child("Sincronismo").getValue()
70                              .toString();
71                      navUsername.setText(dataSnapshot.child("Apodo").getValue().toString());
72                      correoFront.setText(dataSnapshot.child("Correo").getValue().toString());
73                      DbRef=null;
74                  }
75              }
76              @Override
77              public void onCancelled(@NonNull DatabaseError databaseError) {
78              }
79          });
80          actionBarDrawerToggle2=new ActionBarDrawerToggle( activity: this,drawerLayout2,toolbar2
81                  ,"open2","close2");
82          drawerLayout2.addDrawerListener(actionBarDrawerToggle2);
83          actionBarDrawerToggle2.setDrawerIndicatorEnabled(true);
84          actionBarDrawerToggle2.syncState();
85          //establecer el evento onclick al navigationview
86          navigationView2.setNavigationItemSelectedListener(this);
87          //cargar fragment inicial
88          fragmentManager2=getSupportFragmentManager();
89          fragmentTransaction2=fragmentManager2.beginTransaction();
90          fragmentTransaction2.add(R.id.containerFamiliar,new FamiliarFragment());
91          fragmentTransaction2.commit();
92          txtViewHea2=(TextView) findViewById(R.id.textView4);
```

Código 2.42. Inicialización de elementos menú en ActivityFamiliar

- **Código para redirigir a ítem "Ubicación"**

Gracias a *onNavigationItemSelected* es posible realizar este proceso, la línea 126 permitirá que el menú se guarde al momento de seleccionar un ítem, la línea 127 hace referencia al ítem seleccionado y verifica la selección; en la línea 128 ayuda a saber si el usuario Familiar ya tiene un sincronismo con un usuario Principal y la línea 134 se reemplaza a la actividad *MapaActivityFamiliar()* como se indica en el código 2.43.

```
124 0   public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
125
126         drawerLayout2.closeDrawer(GravityCompat.START);
127         if (menuItem.getItemId()==R.id.Ubicacion){
128             if(condicionSincronismoPrincipal.equals("0")){
129
130                 Toast.makeText( context: this,  text: "Por Favor, pide sincronizar a usuario Principal", Toast.LENGTH_LONG).show();
131
132             }else {
133
134                 startActivity(new Intent( packageContext: ActivityFamiliar.this,MapaActivityFamiliar.class));
135
136             }
137
138         }
```

Código 2.43. Redirigir a ítem “Ubicación” en ActivityFamiliar

- **Código para redirigir a ítem “Estadística”, “Chat” y “Cerrar Sesión”**

Las líneas 139, 161 y 164 hace referencia al ítem seleccionado y verifica la selección; de la línea 145 a 148 se reemplaza el fragment con *ReporteFamiliarFragment(idUsuarioPrincipal)*, de la línea 158 a 161 se reemplaza el fragment con *ChatFragmentFamiliar()* y la línea 166 permite cerrar sesión como se observa en el código 2.44.

```
139         if (menuItem.getItemId()==R.id.Estadistica){
140             if(condicionSincronismoPrincipal.equals("0")){
141
142                 Toast.makeText( context: this,  text: "Por Favor, pide sincronizar a usuario Principal", Toast.LENGTH_LONG).show();
143
144             }else {
145                 fragmentManager2=getSupportFragmentManager();
146                 fragmentTransaction2=fragmentManager2.beginTransaction();
147                 fragmentTransaction2.replace(R.id.containerFamiliar,new ReporteFamiliarFragment(condicionSincronismoPrincipal));
148                 fragmentTransaction2.commit();
149             }
150         }
151         if (menuItem.getItemId()==R.id.Chat){
152             if(condicionSincronismoPrincipal.equals("0")){
153
154                 Toast.makeText( context: this,  text: "Por Favor, pide sincronizar a usuario Principal", Toast.LENGTH_LONG).show();
155
156             }else {
157
158                 fragmentManager2=getSupportFragmentManager();
159                 fragmentTransaction2=fragmentManager2.beginTransaction();
160                 fragmentTransaction2.replace(R.id.containerFamiliar,new ChatFragmentFamiliar());
161                 fragmentTransaction2.commit();
162             }
163         }
164         if (menuItem.getItemId()==R.id.CerrarSesion){
165             DbRef=null;
166             baseAutenticacion.signOut();
167
168             startActivity(new Intent( packageContext: ActivityFamiliar.this,MainActivity.class));
169             finish();
170         }
171         return false;
```

Código 2.44. Redirigir a ítem en ActivityFamiliar

- **Opción de salir de la aplicación**

Esta opción se efectúa cuando el usuario presione el botón por defecto “atrás” de cada celular independiente de la versión de Android que tenga instalado, de la línea 98 a 113 se creará un diálogo de alerta para confirmar que se desea salir de la aplicación, en caso de confirmar esta acción finalizará la actividad borrando cualquier dato guardado en cache (código 2.45).

```
96  @   public boolean onKeyDown(int keyCode, KeyEvent event) {
97          if(keyCode == event.KEYCODE_BACK){
98              final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder( context: this);
99              confirmacionDialog.setMessage(Html.fromHtml( source: "Si sales PERDERÁS los " +
100                     "datos que aun no hayas guardado,"+"<b>" +" ¿DESEAS SALIR?  "+"</b>")).setCancelable(false)
101                     .setPositiveButton( text: "Si", (dialog, which) -> {
104                             finish();
105                     }).setNegativeButton( text: "No", (dialog, which) -> {
109                     dialog.cancel();
110             });
112             AlertDialog alerta = confirmacionDialog.create();
113             alerta.setTitle("ALERTA");
114             alerta.show();
115         }
116         return super.onKeyDown(keyCode, event);
```

Código 2.45. OnKeyDown RegistroActivity en ActivityFamiliar

2.5.18. VISUALIZAR UBICACIONES (MAPAACTIVITYFAMILIAR)

Es la actividad donde se visualiza las diferentes ubicaciones o recorridos por fecha de donde se localizaba el usuario Principal con el cual se tiene sincronismo.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, cargar lista de marcadores para mapa y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.46.

```
31  public class MapaActivityFamiliar extends FragmentActivity implements OnMapReadyCallback {
32
33      private GoogleMap mMap;
34      private DatabaseReference DbRef;
35      private FirebaseAuth baseAutenticacion;
36      private ArrayList<Marker> tmpRealTimeMarkers=new ArrayList<>();
37      private ArrayList<Marker> realTimeMarkers=new ArrayList<>();
38      double latitudDouble,longitudDouble;
39
```

Código 2.46. Creación de atributos en MapaActivityFamiliar

- **Código para funcionamiento**

En la línea 57 se llama a *onMapReady()* el cual acepta como atributo un mapa de Google, en la línea 61 se llama a la base de datos para encontrar los diferentes puntos del usuario Principal con el que se tiene sincronismo; de la línea 70 a 72 se limpia el array de marcadores que se utilizara como fuente, es decir, los datos de este elemento se setearán y permitirá evitar redundancia. De la línea 73 a 84 se obtiene la fecha, longitud y latitud de un solo punto y se lo guarda en el array de marcadores; de la línea 85 a 95 se moverá la vista del mapa al último punto registrado en la base de datos por parte del usuario Principal como se indica en el código 2.47.

```
57  public void onMapReady(GoogleMap googleMap) {
58      mMap = googleMap;
59
60      final String id=baseAutenticacion.getCurrentUser().getUid();
61      DbRef.child("USUARIOS").child("FAMILIAR").child(id).addListenerForSingleValueEvent(new ValueEventListener() {
62          @Override
63          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
64              if(dataSnapshot.exists()){}
65              final String idPrincipal=dataSnapshot.child("Sincronismo").getValue().toString();
66
67              DbRef.child("UBICACION").child(idPrincipal).addListenerForSingleValueEvent(new ValueEventListener() {
68                  @Override
69                  public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
70                      for (Marker marker:realTimeMarkers){
71                          marker.remove();
72                      }
73                      for (DataSnapshot snapshot:dataSnapshot.getChildren()) {
74                          PuntosMapa pm = snapshot.getValue(PuntosMapa.class);
75                          String Fecha=pm.getFecha();
76                          String latitud=pm.getLatitud();
77                          String longitud=pm.getLongitud();
78                          latitudDouble=Double.parseDouble(latitud);
79                          longitudDouble=Double.parseDouble(longitud);
80                          MarkerOptions markerOptions=new MarkerOptions();
81                          markerOptions.title(Fecha);
82                          markerOptions.position(new LatLng(latitudDouble,longitudDouble));
83                          tmpRealTimeMarkers.add(mMap.addMarker(markerOptions));
84                      }
85                      realTimeMarkers.clear();
86                      realTimeMarkers.addAll(tmpRealTimeMarkers);
87                      LatLng miUbicacion = new LatLng(latitudDouble, longitudDouble);
88                      mMap.moveCamera(CameraUpdateFactory.newLatLng(miUbicacion));
89                      CameraPosition cameraPosition = new CameraPosition.Builder()
90                              .target(miUbicacion)
91                              .zoom(18)
92                              .bearing(90)
93                              .tilt(45)
94                              .build();
95                      mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
96
97                  }
```

Código 2.47. Funcionamiento de MapaActivityFamiliar

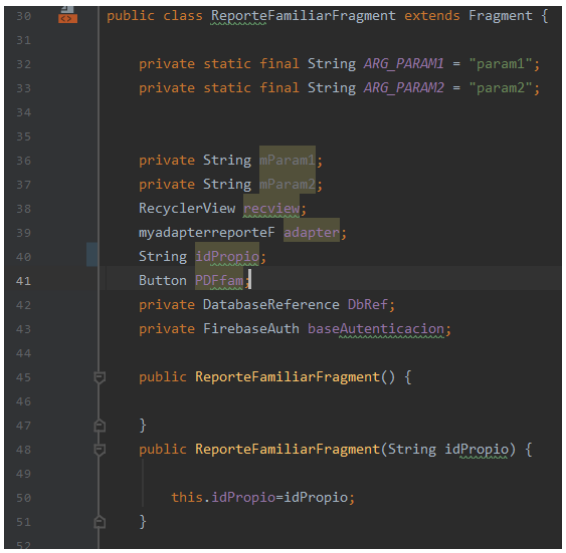
2.5.19. REPORTE TRANSCRIPCIONES (REPORTEFAMILIARFRAGMENT)

Es el fragment donde se visualiza la información básica de todas las transcripciones del usuario Principal con el que se tiene sincronismo y se puede descargar un PDF con la información anterior.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese Fragment.

- **Código para creación de atributos**

Se crea botón para crear el PDF, adaptador, constructor del fragment para recibir el ID del usuario referente a la base de datos y atributos pertenecientes a la base de datos y autenticación como se puede ver en el código 2.48.



```
30  public class ReporteFamiliarFragment extends Fragment {
31
32      private static final String ARG_PARAM1 = "param1";
33      private static final String ARG_PARAM2 = "param2";
34
35
36      private String mParam1;
37      private String mParam2;
38      RecyclerView recview;
39      myadapterreporteF adapter;
40      String idPropio;
41      Button PDFfam;
42      private DatabaseReference DbRef;
43      private FirebaseAuth baseAutenticacion;
44
45      public ReporteFamiliarFragment() {
46
47      }
48      public ReporteFamiliarFragment(String idPropio) {
49
50          this.idPropio=idPropio;
51      }
52
```

Código 2.48. Creación de atributos en ReporteFamiliarFragment

- **Código para funcionamiento**

De la línea 75 a 79 se inicializa todos los atributos, de la línea 80 a 86 se tiene la acción cuando se pulsa el botón para obtener el archivo PDF llamando a *PruebaFamiliar();* la función *obtener()* llamará a la base de datos para obtener todas las transcripciones y mostrarlas en pantalla (código 2.49).

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view= inflater.inflate(R.layout.fragment_reporte_familiar, container,  attachToRoot: false);
        recview=(RecyclerView)view.findViewById(R.id.recvwRepFam);
        baseAutenticacion= FirebaseAuth.getInstance();
        final String id=baseAutenticacion.getCurrentUser().getUid();
        DbRef= FirebaseDatabase.getInstance().getReference();
        PDFfam=(Button)view.findViewById(R.id.butReporFam);
        PDFfam.setOnClickListener((v) -> {
                Intent i = new Intent(getActivity(), PruebaFamiliar.class);
                i.putExtra( name: "dato",idPropio);
                startActivity(i);
        });
        obtener();

        return view;
    }
public void obtener(){
    final String id=baseAutenticacion.getCurrentUser().getUid();
    recview.setLayoutManager(new LinearLayoutManager(getContext()));
    FirebaseRecyclerOptions<modelrepfam> options =
            new FirebaseRecyclerOptions.Builder<modelrepfam>()
                    .setQuery(DbRef.child("TRANSCRIPCIONES").child(idPropio), modelrepfam.class)
                    .build();
    adapter=new myadapterreporteF(options);
    recview.setAdapter(adapter);
}
```

Código 2.49. Funcionamiento de ReporteFamiliarFragment

2.5.20. PRUEBA FAMILIAR

Es la actividad donde se crea la estructura del PDF, se obtiene datos de la base de datos y se crea un tiempo de espera para evitar errores ya que la base de datos funciona en tiempo real. También el mismo formato que se tiene en esta clase se utilizará para crear PDF en usuario Principal.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese Fragment.

- **Código para creación de atributos**

Se crea nombres que tomará la carpeta y el documento PDF, documentos, archivos, tablas, contadores, formato de letras y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.50.

```
41  public class PruebaFamiliar extends AppCompatActivity {
42      int contador;
43      private DatabaseReference DbRef;
44      private FirebaseAuth baseAutenticacion;
45      private Font fNegrita=new Font(Font.TIMES_ROMAN, size: 30,Font.BOLD);
46      private Font fAzul=new Font(Font.TIMES_ROMAN, size: 20,Font.BOLD, Color.BLUE);
47      String NOMBRE_DIRECTORIO = "MisPdfsFam";
48      String NOMBRE_DOCUMENTO = "Transcripcion.pdf";
49      String nombreUsuario;
50      Button btnGenerar;
51      Document documento;
52      PdfPTable tabla;
53      File file;
54      FileOutputStream ficheroPDF;
55      PdfWriter writer;
56      String dato;
```

Código 2.50. Creación de atributos en PruebaFamiliar

- **Código para funcionamiento**

De la línea 64 a 68 se inicializa los atributos, en la línea 70 se llama a la base de datos, en la línea 75 se obtiene el total de transcripciones en número y en la línea 76 se obtiene el nombre del usuario al que pertenece el PDF como se indica en el código 2.51.

```
64      dato=getIntent().getStringExtra( name: "dato");
65
66      baseAutenticacion= FirebaseAuth.getInstance();
67      DbRef= FirebaseDatabase.getInstance().getReference();
68      final String idPropio=baseAutenticacion.getCurrentUser().getUid();
69
70      DbRef.addValueEventListener(new ValueEventListener() {
71          @Override
72          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
73              if(dataSnapshot.exists()){
74
75                  contador=(int)dataSnapshot.child("TRANSCRIPCIONES").child(dato).getChildrenCount();
76                  nombreUsuario=dataSnapshot.child("USUARIOS").child("PRINCIPAL").child(dato).child("Apodo").getValue().toString();
77              }
78          }
79
80          @Override
81          public void onCancelled(@NonNull DatabaseError databaseError) {
82
83
84          }
85      });
```

Código 2.51. Funcionamiento de PruebaFamiliar

- **Permisos y tabla**

De la línea 87 a 99 se llama a la base de datos para obtener cada transcripción y se guarda en *tabla* que es una tabla del formato *documento* creado anteriormente. De la línea 103 a 109

se piden permisos para acceder al almacenamiento interno de memoria. En la línea 111 inicia la acción resultante de pulsar el btnGenerar como se puede ver en el código 2.52.

```
87      DbRef.child("TRANSCRIPCIONES").child(dato).addValueEventListener(new ValueEventListener() {
88          @Override
89          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
90              tabla = new PdfPTable( numColumns: 1);
91              for (DataSnapshot snapshot:dataSnapshot.getChildren()) {
92                  tabla.addCell( text: "\t\t\t\t\t\t\t\t"+snapshot.child("titulo").getValue().toString() +
93                          ":\n\t\t\t\t\t\t\t\t"+snapshot.child("fecha").getValue().toString());
94              }
95          }
96          @Override
97          public void onCancelled(@NonNull DatabaseError databaseError) {
98          }
99      });
100     //boton
101     btnGenerar = findViewById(R.id.btnGenerarFa);
102     // Permisos
103     if(ActivityCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
104             PackageManager.PERMISSION_GRANTED &&
105             ActivityCompat.checkSelfPermission(this, Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
106                     PackageManager.PERMISSION_GRANTED) {
107         ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,},
108                 requestCode: 1000);
109     }
110
111     btnGenerar.setOnClickListener((v) -> {
114             crearPDF2();
115             Toast.makeText( context: PruebaFamiliar.this, text: "SE CREO EL PDF", Toast.LENGTH_LONG).show();
116             finish();
117     });
```

Código 2.52. Permisos y tabla de PDF.

- **Código para función crearPDF2()**

Se inicializa los objetos como archivo, fichero, formato en PDF; se abre el documento, se guarda los diferentes párrafos con el formato de letra correspondiente, se guarda la tabla obtenida en la función anteriormente y se cierra el documento como se observa en el código 2.53.

```
131     public void crearPDF2(){
132         documento= new Document();
133         try {
134             file=crearFichero2(NOMBRE_DOCUMENTO);
135             ficheroPDF=new FileOutputStream(file.getAbsolutePath());
136
137             writer=PdfWriter.getInstance(documento,ficheroPDF);
138             documento.open();
139             Paragraph paragraph=new Paragraph( string: "Hola la tabla es de: "+nombreUsuario,fNegrita);
140             paragraph.setAlignment(Element.ALIGN_CENTER);
141             documento.add(paragraph);
142             Paragraph paragraph2=new Paragraph( string: "Tiene un total de: "+contador+" transcripciones\n\n",fAzul);
143             paragraph2.setAlignment(Element.ALIGN_CENTER);
144             documento.add(paragraph2);
145             documento.add(tabla);
146         }catch (DocumentException e){
147
148         }catch (IOException e){
149
150         }finally {
151             documento.close();
152         }
153     }
```

Código 2.53. Función crearPDF2() de PruebaFamiliar

- **Código para función *getRuta()* y *crarFichero2()***

De la línea 168 a 182 se creará la ruta en el dispositivo donde se almacenará el documento PDF. De la línea 160 a 167 se creará el fichero ya dentro de la memoria del dispositivo para ser editado con el proyecto (código 2.54).

```
160     public File crearFichero2(String nombreFichero){
161         File ruta=getRuta();
162         File fichero=null;
163         if (ruta!=null){
164             fichero=new File(ruta,nombreFichero);
165         }
166         return fichero;
167     }
168     public File getRuta(){
169         File ruta=null;
170         if(Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())){
171             ruta=new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),NOMBRE_DIRECTORIO);
172             if(ruta!=null){
173                 if (!ruta.mkdir()){
174                     if (!ruta.exists()){
175                         return null;
176                     }
177                 }
178             }
179         }
180         return ruta;
181     }
182 }
183
```

Código 2.54. Función getRuta() y crarFichero2() de PruebaFamiliar

2.5.21. MENSAJEAR (CHATFRAGMENTFAMILIAR)

Es la actividad donde se crea la estructura para recibir o enviar mensajes entre el usuario Familiar y su usuario Principal con quien tiene sincronismo. También el mismo formato que se tiene en esta clase se utilizará para crear *ChatFragment* que es enviar y recibir mensajes desde el usuario Principal.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese Fragment.

- **Código para creación de atributos**

Se crea el adaptador, los atributos para guardar el mensaje a escribir, el botón para enviar el mensaje y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.55.

```
34  public class ChatFragmentFamiliar extends Fragment {
35      // TODO: Rename parameter arguments, choose names that match
36      // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
37      private static final String ARG_PARAM1 = "param1";
38      private static final String ARG_PARAM2 = "param2";
39
40      // TODO: Rename and change types of parameters
41      private String mParam1;
42      private String mParam2;
43      RecyclerView recview;
44      myadapterchat adapter;
45      String idPrincipal,oracionText;
46      TextView oracion;
47
48      private DatabaseReference DbRef;
49      private FirebaseAuth baseAutenticacion;
50      private ImageButton btnEnviarFamiliar;
51
52      public ChatFragmentFamiliar() {
53
54      }
```

Código 2.55. Creación de atributos en ChatFragmentFamiliar

- **Código para funcionamiento**

De la línea 78 a 84 se inicializa los atributos, de la línea 86 a 91 se captura los mensajes enviados y recibidos guardados en la base de datos y se muestra en pantalla; al dar clic en al botón para enviar mensaje se llama a la función *enviarMensaje()* como se observa en el código 2.56.

```
76  @   public View onCreateView(LayoutInflater inflater, ViewGroup container,
77                               Bundle savedInstanceState) {
78          baseAutenticacion= FirebaseAuth.getInstance();
79          final String id=baseAutenticacion.getCurrentUser().getUid();
80          DbRef= FirebaseDatabase.getInstance().getReference();
81
82          View view= inflater.inflate(R.layout.fragment_chat_familiar, container,  attachToRoot: false);
83          recview=(RecyclerView)view.findViewById(R.id.recviewChatFamiliar);
84          recview.setLayoutManager(new LinearLayoutManager(getContext()));
85
86          FirebaseRecyclerOptions<modelChat> options =
87                  new FirebaseRecyclerOptions.Builder<modelChat>()
88                          .setQuery(DbRef.child("MENSAJES").child(id), modelChat.class)
89                          .build();
90          adapter=new myadapterchat(options);
91          recview.setAdapter(adapter);
92          oracion=(TextView)view.findViewById(R.id.editTextOracionFamiliar);
93          btnEnviarFamiliar=(ImageButton)view.findViewById(R.id.botonEnviarFamiliar);
94
95          btnEnviarFamiliar.setOnClickListener((v) -> {
98                  oracionText=oracion.getText().toString();
99                  if (!oracionText.isEmpty()) {
100                     enviarMensaje();
101
102                 }else{
103                     Toast.makeText(getActivity(),  text: "Escribe tu mensaje", Toast.LENGTH_SHORT).show();
104                 }
105         });
107         return view;
```

Código 2.56. Funcionamiento de ChatFragmentFamiliar

- **Código de función *enviarMensaje()***

En la línea 111 se llama a la base de datos y se apunta al nodo que contiene los mensajes del usuario, de la línea 115 a la 124 se guarda los mensajes que se envíe en la base de datos tanto para el usuario que envía como para el que recibe como se indica en el código 2.57.

```
109     public void enviarMensaje(){
110         final String id=baseAutenticacion.getCurrentUser().getUid();
111         DbRef.child("USUARIOS").child("FAMILIAR").child(id).addListenerForSingleValueEvent(new ValueEventListener() {
112             @Override
113 O1          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
114                 if(dataSnapshot.exists()){
115                     idPrincipal=dataSnapshot.child("Sincronismo").getValue().toString();
116                     final Map<String,Object> mapUsuario=new HashMap<>();
117                     mapUsuario.put( k: "estado", v: "recibido");
118                     mapUsuario.put( k: "texto",oracionText);
119                     DbRef.child("MENSAJES").child(idPrincipal).push().setValue(mapUsuario);
120                     final Map<String,Object> mapUsuario2=new HashMap<>();
121                     mapUsuario2.put( k: "estado", v: "enviado");
122                     mapUsuario2.put( k: "texto",oracionText);
123                     DbRef.child("MENSAJES").child(id).push().setValue(mapUsuario2);
124                     oracion.setText("");
125
126                 }
127             }
```

Código 2.57. Función enviarMensaje() de ChatFragmentFamiliar

2.5.22. MENÚ PRINCIPAL (ACTIVITYTRANSCRIPCION)

Es la actividad donde se programará el menú desplegable perteneciente al usuario Principal y contendrá el llamado a cada fragment al seleccionar cada ítem del menú.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, cargar el menú principal completo y atributos pertenecientes a la base de datos y autenticación como se puede ver en el código 2.58.

```
39  public class ActivityTranscripcion extends AppCompatActivity implements NavigationView.OnNavigationItemSelectedListener{
40      // private Button buttonCerrar;
41      //private FirebaseAuth authUserAT;
42      DrawerLayout drawerLayout;
43      ActionBarDrawerToggle actionBarDrawerToggle;
44      Toolbar toolbar;
45      NavigationView navigationView;
46      //variables para cargar el fragment
47      FragmentManager fragmentManager;
48      FragmentTransaction fragmentTransaction;
49      //text
50      TextView txtViewHea;
51      //base de datos
52      private DatabaseReference DbRef;
53      private FirebaseAuth baseAutenticacion;
54      String condicionSincronismo;
```

Código 2.58. Creación de atributos en ActivityTranscripcion

- **Código para inicialización de elementos y cargar datos de usuario al menú**

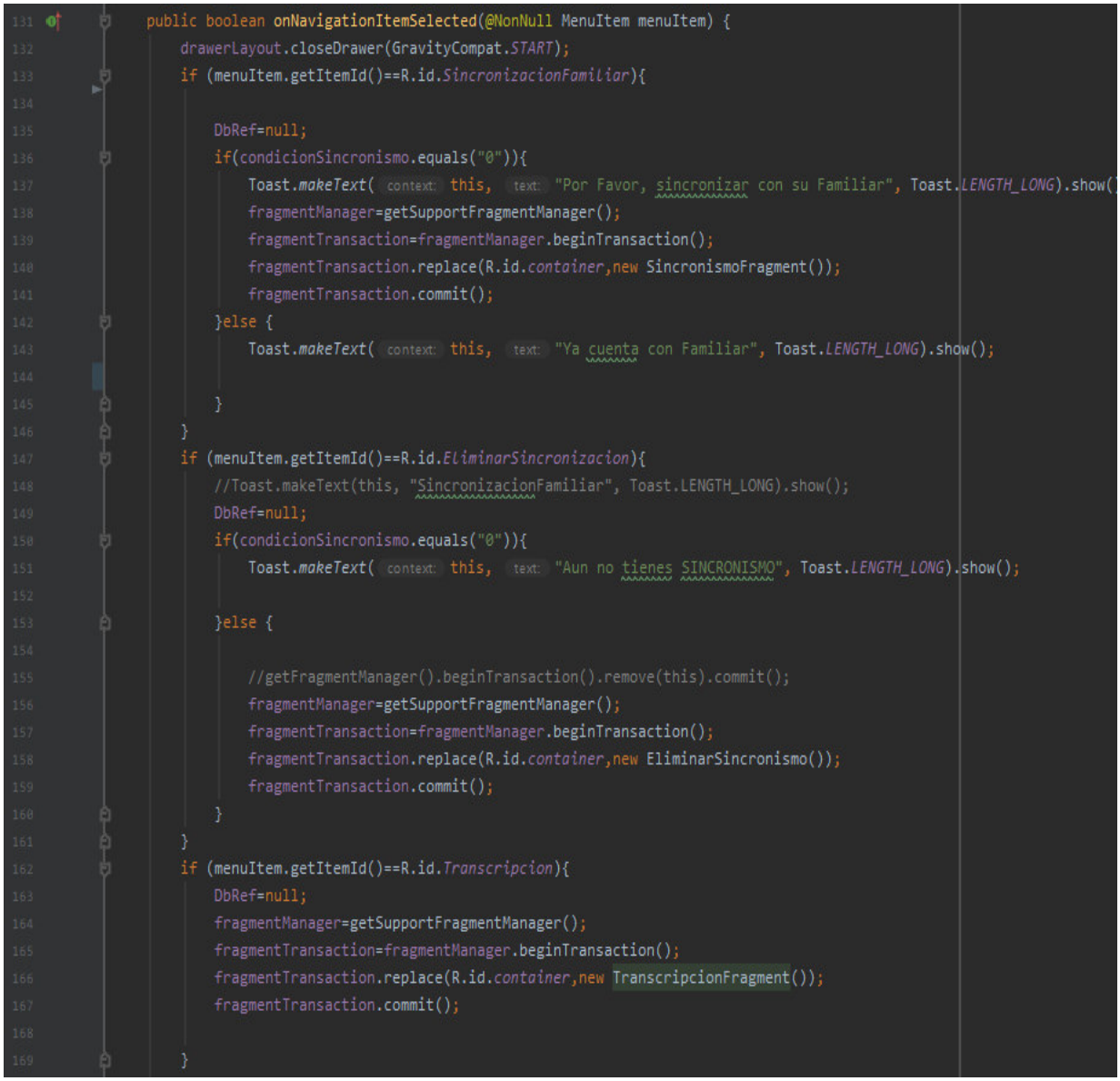
En esta sección se carga los datos correspondientes al usuario haciendo un pedido a la base de datos con la línea de código 72, de la línea 90 a la 93 se carga el menú, la línea 95 permite que los ítems del menú sean seleccionables y de la 99 a la 102 se carga el fragment inicial, en este caso es *TranscripcionFragment ()* como se observa en el código 2.59.

```
70          DbRef.child(id).addValueEventListener(new ValueEventListener() {
71              @Override
72              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
73                  if(dataSnapshot.exists()){
74                      View headerView = navigationView.getHeaderView( index: 0);
75                      TextView navUsername = (TextView) headerView.findViewById(R.id.textView4);
76                      TextView correoFront=(TextView)headerView.findViewById(R.id.textView5);
77                      navUsername.setText(dataSnapshot.child("Apodo").getValue().toString());
78                      correoFront.setText(dataSnapshot.child("Correo").getValue().toString());
79                      condicionSincronismo=dataSnapshot.child("Sincronismo").getValue().toString();
80                      DbRef=null;
81                  }
82              }
83
84              @Override
85              public void onCancelled(@NonNull DatabaseError databaseError) {
86
87              }
88          });
89          //mostrar menu lateral
90          actionBarDrawerToggle=new ActionBarDrawerToggle( activity: this,drawerLayout,toolbar,"open","close");
91          drawerLayout.addDrawerListener(actionBarDrawerToggle);
92          actionBarDrawerToggle.setDrawerIndicatorEnabled(true);
93          actionBarDrawerToggle.syncState();
94          //establecer el evento onclick al navigationview
95          navigationView.setNavigationItemSelectedListener(this);
96
97
98          //cargar fragment principal
99          fragmentManager=getSupportFragmentManager();
100         fragmentTransaction=fragmentManager.beginTransaction();
101         fragmentTransaction.add(R.id.container,new TranscripcionFragment());
102         fragmentTransaction.commit();
103         txtViewHea=(TextView) findViewById(R.id.textView4);
104
105     }
```

Código 2.59. Inicialización de elementos ActivityTranscripcion

- **Código para redirigir a item "SincronizacionFamiliar", "EliminarSincronizacion" "Transcripcion"**

Gracias a *onNavigationItemSelected* es posible realizar este proceso, la línea 132 permitirá que el menú se guarde al momento de seleccionar un ítem, las líneas 136, 147 y 162 hace referencia al ítem seleccionado y verifica la selección; de la línea 138 a 141 se reemplaza el fragment con *SincronismoFragment (),* de la línea 156 a 159 se reemplaza el fragment con *EliminarSincronismo () y* de la línea 164 a 167 se reemplaza el fragment con *TranscripcionFragment ()* como se puede ver en el código 2.60.

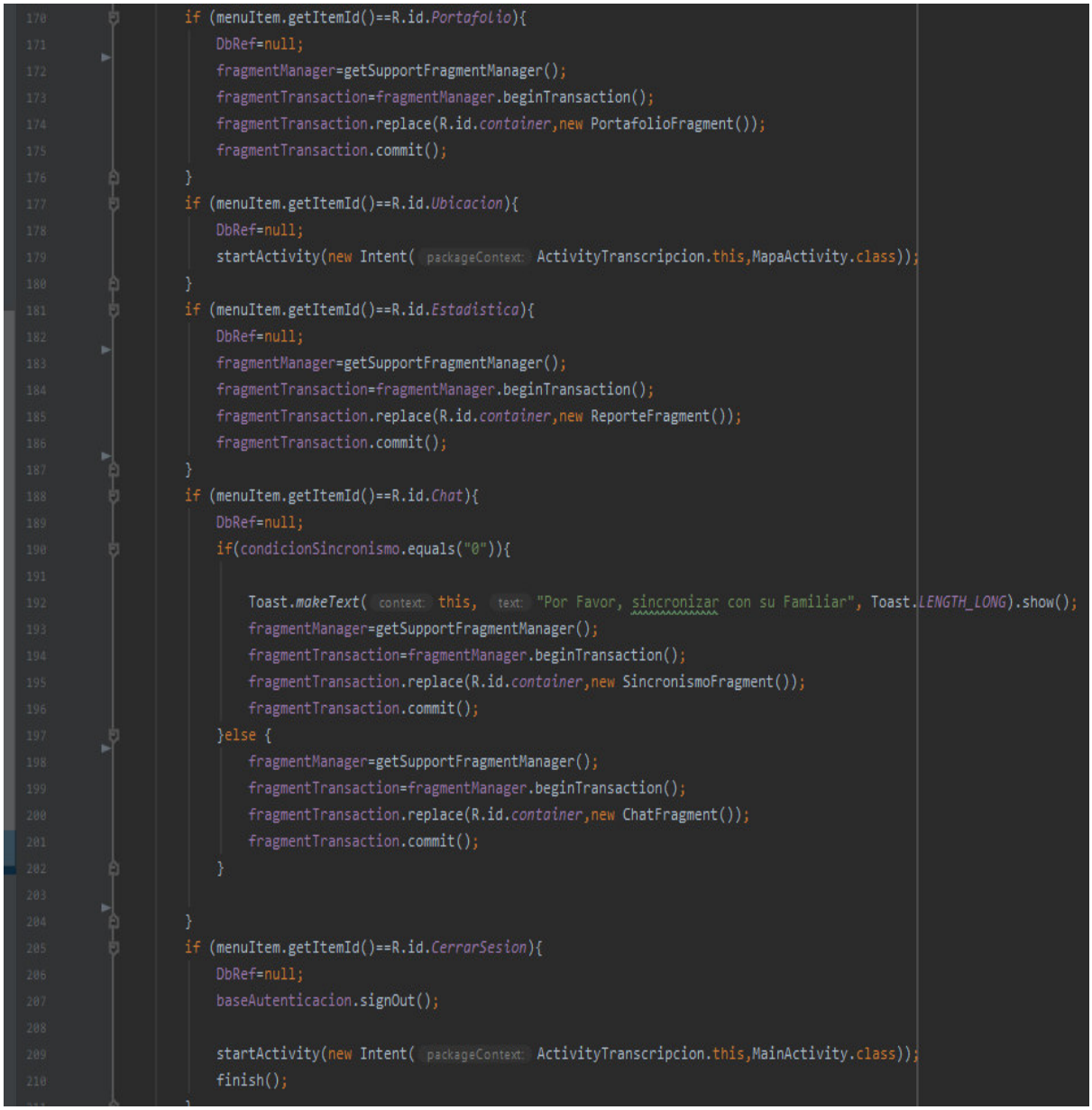


```
131  public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {
132      drawerLayout.closeDrawer(GravityCompat.START);
133      if (menuItem.getItemId()==R.id.SincronizacionFamiliar){
134
135          DbRef=null;
136          if(condicionSincronismo.equals("0")){
137              Toast.makeText( context: this,  text: "Por Favor, sincronizar con su Familiar", Toast.LENGTH_LONG).show(
138              fragmentManager=getSupportFragmentManager();
139              fragmentTransaction=fragmentManager.beginTransaction();
140              fragmentTransaction.replace(R.id.container,new SincronismoFragment());
141              fragmentTransaction.commit();
142          }else {
143              Toast.makeText( context: this,  text: "Ya cuenta con Familiar", Toast.LENGTH_LONG).show();
144
145          }
146      }
147      if (menuItem.getItemId()==R.id.EliminarSincronizacion){
148          //Toast.makeText(this, "SincronizacionFamiliar", Toast.LENGTH_LONG).show();
149          DbRef=null;
150          if(condicionSincronismo.equals("0")){
151              Toast.makeText( context: this,  text: "Aun no tienes SINCRONISMO", Toast.LENGTH_LONG).show();
152
153          }else {
154
155              //getFragmentManager().beginTransaction().remove(this).commit();
156              fragmentManager=getSupportFragmentManager();
157              fragmentTransaction=fragmentManager.beginTransaction();
158              fragmentTransaction.replace(R.id.container,new EliminarSincronismo());
159              fragmentTransaction.commit();
160          }
161      }
162      if (menuItem.getItemId()==R.id.Transcripcion){
163          DbRef=null;
164          fragmentManager=getSupportFragmentManager();
165          fragmentTransaction=fragmentManager.beginTransaction();
166          fragmentTransaction.replace(R.id.container,new TranscripcionFragment());
167          fragmentTransaction.commit();
168
169      }
```

Código 2.60. Redirigir a item en ActivityTranscripcion

- **Código para redirigir a ítem “Portafolio”, “Ubicación”,” Estadística”, “Chat” y “Cerrar Sesión”**

Las líneas 170, 177, 181, 188 y 205 hace referencia al ítem seleccionado y verifica la selección; de la línea 172 a 175 se reemplaza el fragment con *PortafolioFragment ()*, en la línea 179 se llama a la actividad *MapaActivity (),* de la línea 183 a 186 se reemplaza el fragment con *ChatFragmentFamiliar(),* de la línea 193 a 196 se reemplaza el fragment con *SincronismoFragment (),* de la línea 198 a 201 se reemplaza el fragment con *ChatFragment()* y la línea 207 permite cerrar sesión (código 2.61).



```
170         if (menuItem.getItemId()==R.id.Portafolio){
171             DbRef=null;
172             fragmentManager=getSupportFragmentManager();
173             fragmentTransaction=fragmentManager.beginTransaction();
174             fragmentTransaction.replace(R.id.container,new PortafolioFragment());
175             fragmentTransaction.commit();
176         }
177         if (menuItem.getItemId()==R.id.Ubicacion){
178             DbRef=null;
179             startActivity(new Intent( packageContext: ActivityTranscripcion.this,MapaActivity.class));
180         }
181         if (menuItem.getItemId()==R.id.Estadistica){
182             DbRef=null;
183             fragmentManager=getSupportFragmentManager();
184             fragmentTransaction=fragmentManager.beginTransaction();
185             fragmentTransaction.replace(R.id.container,new ReporteFragment());
186             fragmentTransaction.commit();
187         }
188         if (menuItem.getItemId()==R.id.Chat){
189             DbRef=null;
190             if(condicionSincronismo.equals("0")){
191
192                 Toast.makeText( context: this,  text: "Por Favor, sincronizar con su Familiar", Toast.LENGTH_LONG).show();
193                 fragmentManager=getSupportFragmentManager();
194                 fragmentTransaction=fragmentManager.beginTransaction();
195                 fragmentTransaction.replace(R.id.container,new SincronismoFragment());
196                 fragmentTransaction.commit();
197             }else {
198                 fragmentManager=getSupportFragmentManager();
199                 fragmentTransaction=fragmentManager.beginTransaction();
200                 fragmentTransaction.replace(R.id.container,new ChatFragment());
201                 fragmentTransaction.commit();
202             }
203
204         }
205         if (menuItem.getItemId()==R.id.CerrarSesion){
206             DbRef=null;
207             baseAutenticacion.signOut();
208
209             startActivity(new Intent( packageContext: ActivityTranscripcion.this,MainActivity.class));
210             finish();
```

Código 2.61. Redirigir a ítems “Chat”

- **Opción de salir de la aplicación**

Esta opción se efectúa cuando el usuario presione el botón por defecto “atrás” de cada celular independiente de la versión de Android que tenga instalado, de la línea 108 a 128 se creará un diálogo de alerta para confirmar que se desea salir de la aplicación, en caso de confirmar esta acción finalizará la actividad borrando cualquier dato guardado en cache como se indica en el código 2.62.

```
106     @Override
107 @   public boolean onKeyDown(int keyCode, KeyEvent event) {
108         if(keyCode == event.KEYCODE_BACK){
109             final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder( context: this);
110             confirmacionDialog.setMessage(Html.fromHtml( source: "Si sales PERDERÁS los datos que a" +
111                     "un no hayas guardado,"+"<b>" +" ¿DESEAS SALIR?  "+"</b>")).setCancelable(false)
112                     .setPositiveButton( text: "Si", (dialog, which) -> {
115                             finish();
116                     }).setNegativeButton( text: "No", (dialog, which) -> {
120                     dialog.cancel();
121             });
123             AlertDialog alerta = confirmacionDialog.create();
124             alerta.setTitle("ALERTA");
125             alerta.show();
126         }
127         return super.onKeyDown(keyCode, event);
128     }
```

Código 2.62. OnKeyDown en ActivityTranscripcion

2.5.23. ENLACE DE USUARIOS (SINCRONISMOFRAGMENT)

Es el fragment donde el usuario Principal puede seleccionar tener un enlace (sincronismo) con un usuario Familiar.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese fragment.

- **Código para creación de atributos**

Se crea adaptador y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.63.

```
20  public class SincronismoFragment extends Fragment {
21
22      private static final String ARG_PARAM1 = "param1";
23      private static final String ARG_PARAM2 = "param2";
24
25
26      private String mParam1;
27      private String mParam2;
28      RecyclerView recview;
29      myadaptersincronismo adapter;
30      private DatabaseReference DbRef;
31      private FirebaseAuth baseAutenticacion;
32
33      public SincronismoFragment() {
34
35      }
```

Código 2.63. Creación de atributos en SincronismoFragment

- **Código para funcionamiento**

De la línea 60 a 66 se inicializa los atributos, de la línea 68 a 74 se hará un llamado a la base de datos apuntando al nodo “FAMILIAR”, se obtendrá todos los usuarios del rol Familiar y se

los mostrará con una lista en pantalla utilizando el adaptador correspondiente como se puede ver en el código 2.64.

```
57  public View onCreateView(LayoutInflater inflater, ViewGroup container,
58                           Bundle savedInstanceState) {
59
60      View view=inflater.inflate(R.layout.fragment_sincronismo, container,  attachToRoot: false);
61      recview=(RecyclerView)view.findViewById(R.id.recviewSincronismo);
62      recview.setLayoutManager(new LinearLayoutManager(getContext()));
63
64      baseAutenticacion= FirebaseAuth.getInstance();
65      final String id=baseAutenticacion.getCurrentUser().getUid();
66      DbRef= FirebaseDatabase.getInstance().getReference().child("USUARIOS");
67
68      FirebaseRecyclerOptions<modelSincronismo> options =
69              new FirebaseRecyclerOptions.Builder<modelSincronismo>()
70                      .setQuery(DbRef.child("FAMILIAR"), modelSincronismo.class)
71                      .build();
72
73      adapter=new myadaptersincronismo(options);
74      recview.setAdapter(adapter);
75
76
77
78      return view;
```

Código 2.64. Funcionamiento de SincronismoFragment

2.5.24. ELIMINAR ENLACE (QUITARSINCRONISMOFRAGMENT)

Es el fragment donde el usuario Principal luego de seleccionar su sincronismo, modificará atributos en la base de datos y se activará las funciones que dependen de tener sincronismo.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese fragment.

- **Código para creación de atributos**

Se crea variables para ID propio y del usuario sincronizado y atributos pertenecientes a la base de datos y autenticación. Se crea un constructor de este fragment ya que al seleccionar el usuario en la parte anterior se necesita enviar el ID para hacer diferentes búsquedas en la base de datos como se observa en el código 2.65.

```java
public class QuitarSincronismoFragment extends Fragment {

    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    private String mParam1;
    private String mParam2;
    String idDeSeleccion;
    private String idSincSeleccion;
    private DatabaseReference DbRef;
    private FirebaseAuth baseAutenticacion;

    public QuitarSincronismoFragment() {
    }
    public QuitarSincronismoFragment(String idDeSeleccion) {
        this.idDeSeleccion=idDeSeleccion;
    }
```

Código 2.65. Creación de atributos en QuitarSincronismoFragment

- **Código para funcionamiento**

De la línea 60 a 66 se inicializa los atributos, en la 72 se hará un llamado a la base de datos apuntando al nodo “FAMILIAR”, luego de la línea 74 a 94 se confirma si el usuario seleccionado ya tiene sincronismo, al no tener se llama a la función *sincronizarUsuarios()* y si ya tiene sincronismo se indica con texto como se indica en el código 2.66.

```java
baseAutenticacion= FirebaseAuth.getInstance();
final String id=baseAutenticacion.getCurrentUser().getUid();
DbRef= FirebaseDatabase.getInstance().getReference().child("USUARIOS");
//btnRegresar=(ImageButton) view.findViewById(R.id.botonRegresarSincronismo);
DbRef.child("FAMILIAR").child(idDeSeleccion).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if(dataSnapshot.exists()){
            idSincSeleccion=dataSnapshot.child("Sincronismo").getValue().toString();
            if(idSincSeleccion.equals("0")){
                sincronizarUsuarios();
            }else if (idSincSeleccion.equals(id)){
                Toast.makeText(getActivity(), text: "Se SINCRONIZÓ correctamente ", Toast.LENGTH_LONG).show();
                DbRef=null;
                AppCompatActivity activity=(AppCompatActivity)getContext();
                activity.getSupportFragmentManager().beginTransaction().replace(R.id.container,new
                        TranscripcionFragment()).addToBackStack(null).commit();
            }else {
                Toast.makeText(getActivity(), text: "El usuario NO PUEDE ser seleccionado ",
                        Toast.LENGTH_LONG).show();
                DbRef=null;
                AppCompatActivity activity=(AppCompatActivity)getContext();
                activity.getSupportFragmentManager().beginTransaction().replace(R.id.container,new
                        SincronismoFragment()).addToBackStack(null).commit();
```

Código 2.66. Funcionamiento de QuitarSincronismoFragment

- **Código de función *sincronizarUsuarios ()***

Esta función modifica los atributos de sincronismo tanto en el usuario Principal como en el Familiar como se observa en el código 2.67.

```
public void sincronizarUsuarios(){
    final String id=baseAutenticacion.getCurrentUser().getUid();

    DbRef.child("FAMILIAR").child(idDeSeleccion).child("Sincronismo").setValue(id);
    DbRef.child("PRINCIPAL").child(id).child("Sincronismo").setValue(idDeSeleccion);
    DbRef=null;
    AppCompatActivity activity=(AppCompatActivity)getContext();
    activity.getSupportFragmentManager().beginTransaction().replace(R.id.container,
            new TranscripcionFragment()).addToBackStack(null).commit();
```

Código 2.67. Función sincronizarUsuarios ()

2.5.25. ELIMINAR ENLACE (ELIMINARSINCRONISMO)

Es el fragment donde el usuario Principal puede eliminar su sincronismo, se modificará atributos en la base de datos y se desactivará las funciones que dependen de tener sincronismo.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese fragment.

- **Código para creación de atributos**

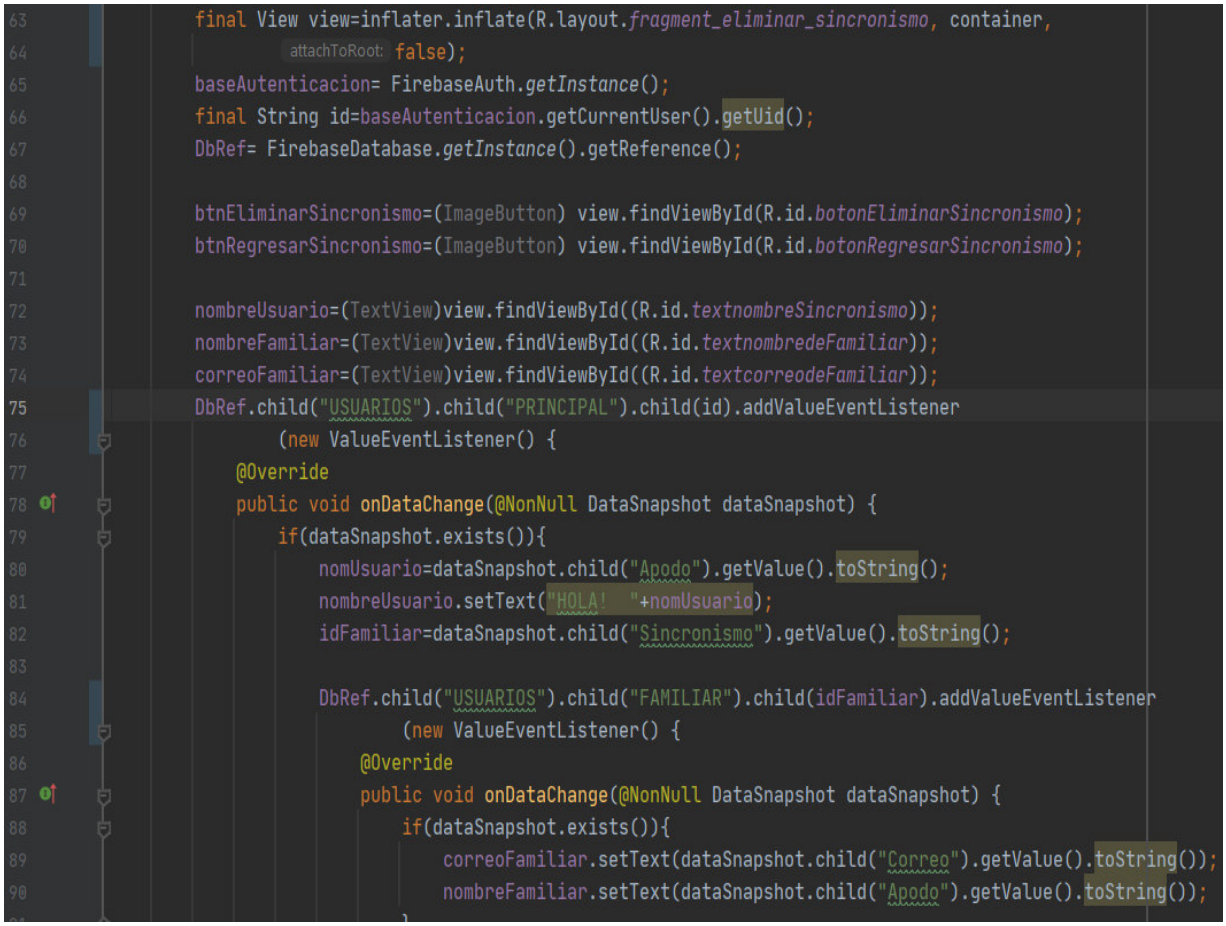
Se crea variables para ID propio y del usuario sincronizado, también para informar cual es el usuario actual que está sincronizado y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.68.

```
29  public class EliminarSincronismo extends Fragment {
30      private static final String ARG_PARAM1 = "param1";
31      private static final String ARG_PARAM2 = "param2";
32
33      private String mParam1;
34      private String mParam2;
35      String idFamiliar,nomUsuario;
36      TextView nombreUsuario,nombreFamiliar,correoFamiliar;
37      private DatabaseReference DbRef;
38      private FirebaseAuth baseAutenticacion;
39      private ImageButton btnEliminarSincronismo,btnRegresarSincronismo;
40      public EliminarSincronismo() {
41      }
```

Código 2.68. Creación de atributos en EliminarSincronismo

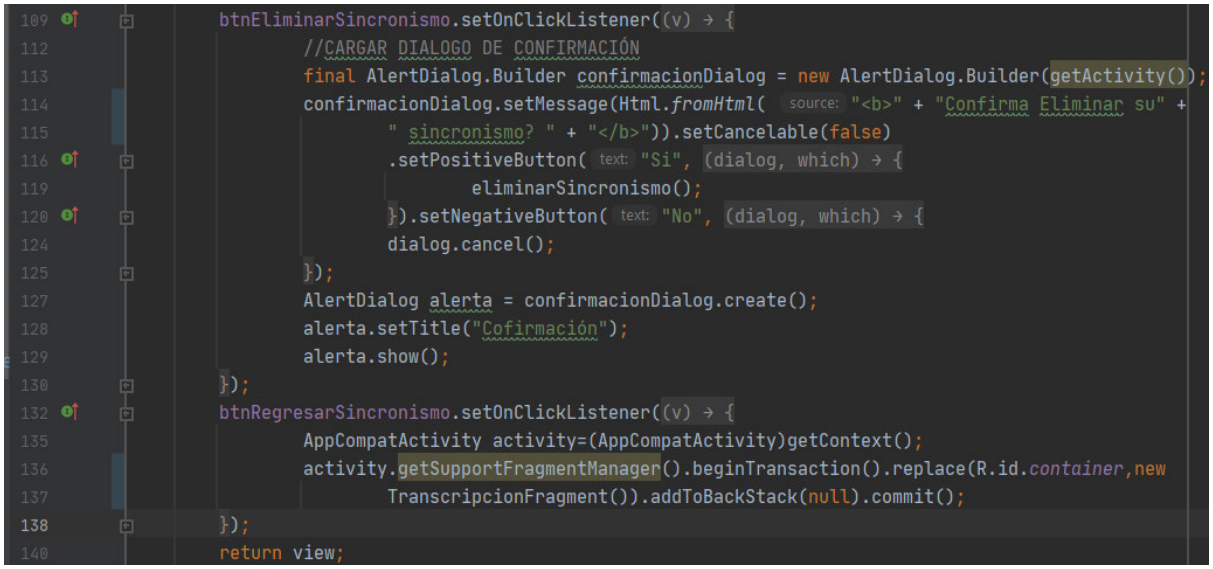
- **Código para funcionamiento**

De la línea 65 a 76 se inicializa los atributos, en la 79 se hará un llamado a la base de datos apuntando al nodo “PRINCIPAL”, luego de la línea 82 a 93 se obtiene información referente al usuario sincronizado desde la base de datos. En la línea 111 se encuentra el método *OnClick* para el botón que eliminará sincronismo, de la línea 115 a la 130 se llamará a un diálogo de alerta que al confirmar enviará hacia *eliminarSincronismo ()* y desde la línea 133 a 148 se regresará a *TranscripcionFragment ()* como se indica en el código 2.69 y 2.70.



```
63          final View view=inflater.inflate(R.layout.fragment_eliminar_sincronismo, container,
64                  attachToRoot: false);
65          baseAutenticacion= FirebaseAuth.getInstance();
66          final String id=baseAutenticacion.getCurrentUser().getUid();
67          DbRef= FirebaseDatabase.getInstance().getReference();
68
69          btnEliminarSincronismo=(ImageButton) view.findViewById(R.id.botonEliminarSincronismo);
70          btnRegresarSincronismo=(ImageButton) view.findViewById(R.id.botonRegresarSincronismo);
71
72          nombreUsuario=(TextView)view.findViewById((R.id.textnombreSincronismo));
73          nombreFamiliar=(TextView)view.findViewById((R.id.textnombredeFamiliar));
74          correoFamiliar=(TextView)view.findViewById((R.id.textcorreodeFamiliar));
75          DbRef.child("USUARIOS").child("PRINCIPAL").child(id).addValueEventListener
76                  (new ValueEventListener() {
77              @Override
78              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
79                  if(dataSnapshot.exists()){
80                      nomUsuario=dataSnapshot.child("Apodo").getValue().toString();
81                      nombreUsuario.setText("HOLA!   "+nomUsuario);
82                      idFamiliar=dataSnapshot.child("Sincronismo").getValue().toString();
83
84                      DbRef.child("USUARIOS").child("FAMILIAR").child(idFamiliar).addValueEventListener
85                              (new ValueEventListener() {
86                          @Override
87                          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
88                              if(dataSnapshot.exists()){
89                                  correoFamiliar.setText(dataSnapshot.child("Correo").getValue().toString());
90                                  nombreFamiliar.setText(dataSnapshot.child("Apodo").getValue().toString());
```

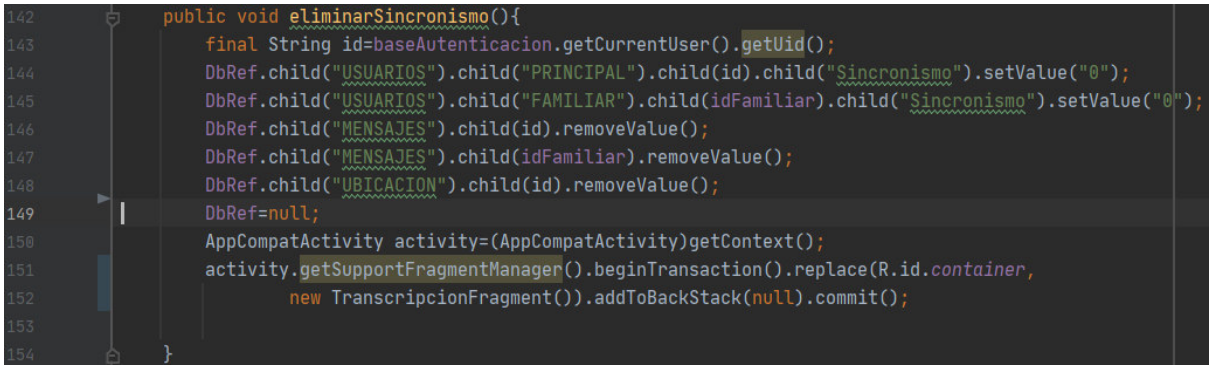
Código 2.69. Funcionamiento de EliminarSincronismo



Código 2.70. Alerta de diálogo en EliminarSincronismo

- **Código de función *eliminarSincronismo ()***

En esta parte se modificará los IDs de sincronismo tanto para el usuario Familiar como para el Principal, se eliminará los mensajes enviados y las ubicaciones anteriores como se puede ver en el código 2.71.



Código 2.71. Función eliminarSincronismo () en EliminarSincronismo

2.5.26. TRANSCRIPCIÓN (TRANSCRPCIONFRAGMENT)

Es el fragment más importante del usuario Principal ya que es el encargado de transcribir oraciones de audio a texto, guardar, pausar, reanudar o descartar la transcripción y aumentar o disminuir el tamaño de la transcripción final.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese fragment.

- **Código para creación de atributos**

Se crea variables para botones, guardar transcripción a medida que se escucha, tamaño de letra inicial, título, temporizador referente a *speech* y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.72.

```
73  public class TranscripcionFragment extends Fragment  {
74
75      private TextView txtTranscripcion;
76      private static final int REQ_CODE_SPEECH_INPUT=100;
77      private String captura="",tituloString;
78      private EditText tituloET;
79      private ImageButton btnHablar, btnGuardar,btnMAs, btnMenos;
80      private View vista;
81      private DatabaseReference DbRef;
82      private FirebaseAuth baseAutenticacion;
83      private Intent intent;
84      private int letraSize=14;
85
```

Código 2.72. Creación de atributos en TranscripcionFragment

- **Código para funcionamiento**

De la línea 92 a 94 se obtiene la instancia de la base de datos en Firebase, se obtiene el id del usuario autenticado y se apunta hacia el nodo “TRANSCRIPCIONES” en la base de datos. En la línea 97 se carga la vista que se enviará al final. En la línea 99 se crea un intent para iniciar la acción de escuchar; ACTION_RECOGNIZE_SPEECH: Inicia una actividad que solicitará al usuario la voz y la enviará a través de un reconocedor de voz. En la línea 101 el reconocedor ajusta los resultados de voz como es el lenguaje. En línea 103 se agrega un lenguaje extra (por seguridad debe ser *getDefault*), en la línea 105 se escribe un mensaje que ayudará al usuario a saber que ya se empezó a transcribir y se está escuchando. La línea 107 ayuda a mantener un tiempo prolongado el mantenerse en “escucha” a la acción speech. De la línea 110 a 115 se inicializa los atributos faltantes como se puede ver en el código 2.73.

```
91          //base de datos
92          baseAutenticacion= FirebaseAuth.getInstance();
93          final String id=baseAutenticacion.getCurrentUser().getUid();
94          DbRef= FirebaseDatabase.getInstance().getReference().child("TRANSCRIPCIONES");
95
96          // Inflate the layout for this fragment
97          vista=inflater.inflate(R.layout.fragment_transcripcion, container,  attachToRoot: false);
98          //intent para inicializar escuchar; ACTION_RECOGNIZE_SPEECH: Inicia una actividad que solicitará a
99          intent=new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);//ACTION_VOICE_SEARCH_HANDS_FREE
100         //El reconocedor utiliza esta información para ajustar los resultados
101         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
102         //Etiqueta de idioma IETF opcional
103         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());
104         //Mensaje de texto opcional para mostrar al usuario cuando le pide que hable
105         intent.putExtra(RecognizerIntent.EXTRA_PROMPT,  value: "...");
106         //
107         intent.putExtra(RecognizerIntent.EXTRA_SPEECH_INPUT_MINIMUM_LENGTH_MILLIS,  value: 0);
108
109
110         txtTranscripcion=(TextView)vista.findViewById(R.id.textoEntrada);
111         btnHablar=(ImageButton) vista.findViewById(R.id.botonHablar);
112         btnGuardar=(ImageButton) vista.findViewById(R.id.botonStop);
113         btnMAs=(ImageButton)vista.findViewById(R.id.botonMas);
114         btnMenos=(ImageButton)vista.findViewById(R.id.botonMenos);
115         tituloET=(EditText)vista.findViewById(R.id.editTextTitulo);
116
```

Código 2.73. Funcionamiento de TranscripcionFragment

- **Código de función *iniciarEntradaVoz ()* y *onActivityResult ()***

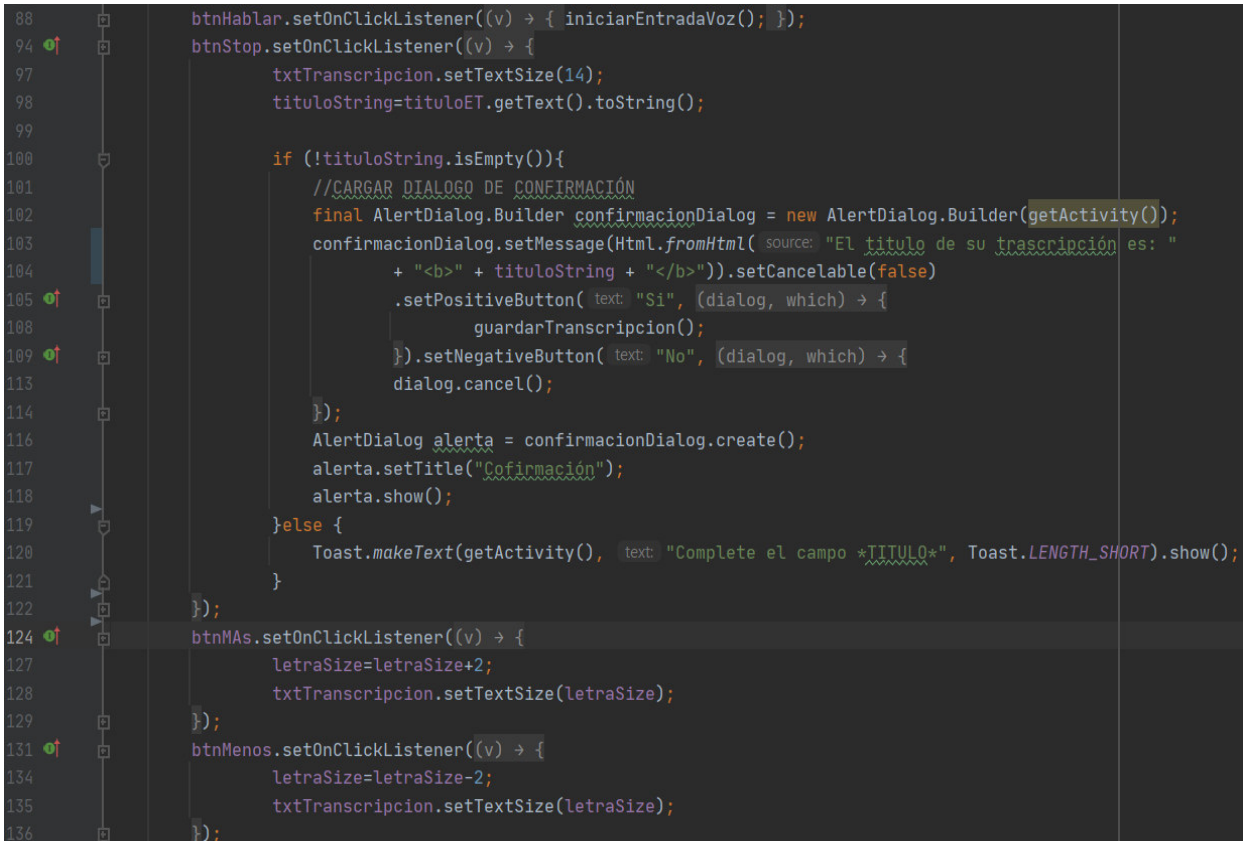
La función *iniciarEntradaVoz ()* situada en las líneas 213 a 218 tiene un enlace con el intent de *RecognizerIntent.ACTION_RECOGNIZE_SPEECH* utilizado para transcripción, que llama a la función *startActivityForResult ()* y acepta como parámetros de entrada al inten y un tiempo de espera; *startActivityForResult ()* activa internamente a la función *onActivityResult ()* la cual se ubica en la línea 220, si cumple las funciones internas como es el tiempo de espera, el resultado sea correcto y la información capturada no sea nula se obtendrá el resultado de la transcripción con *RecognizerIntent.EXTRA_RESULTS* y se guardará en un array de string. De la línea 226 a 236 se guardará los resultados y se concatenará con los anteriores así se puede crear una vista de toda la transcripción. La línea 235 permitirá mantener a la aplicación siempre escuchando y el usuario puede visualizar la transcripción en su dispositivo como se indica en el código 2.74.

```
213     private void iniciarEntradaVoz(){
214
215         try{
216             startActivityForResult(intent,REQ_CODE_SPEECH_INPUT);
217         }catch (ActivityNotFoundException e){}
218     }
219     @Override
220     public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
221         super.onActivityResult(requestCode, resultCode, data);
222         switch (requestCode){
223             case  REQ_CODE_SPEECH_INPUT:{
224                 if(resultCode==RESULT_OK && null!=data){
225                     ArrayList<String> result =data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
226                     if(!captura.isEmpty()){
227                         captura=captura+ ", "+result.get(0);
228                         txtTranscripcion.setText(captura);
229
230                     }else {
231                         captura=captura+result.get(0);
232                         txtTranscripcion.setText(captura);
233                     }
234
235                     iniciarEntradaVoz();
236                 }
237                 break;
238             }
239         }
```

Código 2.74. Función iniciarEntradaVoz ()

- **Código para botones**

Existe cuatro botones principales, la línea 119 llama a escuchar la voz con la función *iniciarEntradaVoz ()*, de la línea 125 a 153 mostrará un diálogo de alerta en el cual si se confirma el título que tendrá la transcripción se llamará a la función *guardarTranscripcion ()*, de la línea 156 a 168 se aumentará y disminuirá el tamaño de letra de la transcripción final como se observa en el código 2.75.



```
88      btnHablar.setOnClickListener((v) → { iniciarEntradaVoz(); });
94      btnStop.setOnClickListener((v) → {
97              txtTranscripcion.setTextSize(14);
98              tituloString=tituloET.getText().toString();
99
100             if (!tituloString.isEmpty()){
101                 //CARGAR DIALOGO DE CONFIRMACIÓN
102                 final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder(getActivity());
103                 confirmacionDialog.setMessage(Html.fromHtml( source: "El titulo de su trascripción es: "
104                         + "<b>" + tituloString + "</b>")).setCancelable(false)
105                         .setPositiveButton( text: "Si", (dialog, which) → {
108                                 guardarTranscripcion();
109                         }).setNegativeButton( text: "No", (dialog, which) → {
113                         dialog.cancel();
114                 });
116                 AlertDialog alerta = confirmacionDialog.create();
117                 alerta.setTitle("Cofirmación");
118                 alerta.show();
119             }else {
120                 Toast.makeText(getActivity(), text: "Complete el campo *TITULO*", Toast.LENGTH_SHORT).show();
121             }
122     });
124     btnMAs.setOnClickListener((v) → {
127             letraSize=letraSize+2;
128             txtTranscripcion.setTextSize(letraSize);
129     });
131     btnMenos.setOnClickListener((v) → {
134             letraSize=letraSize-2;
135             txtTranscripcion.setTextSize(letraSize);
136     });
```

Código 2.75. Botones en TranscripcionFragment

- **Código para función *guardarTranscripcion ()***

De la línea 178 a 187 se apuntará a la base de datos en el nodo “TRANSCRIPCIONES”, se capturará la fecha con el formato día, mes, año y hora; se creará un mapa de atributos con el título, la fecha y la transcripción. De la línea 188 a 201 se guardará en la base de datos si el título ingresado no existe, caso contrario se pedirá que se cambie el título como se puede ver en el código 2.76.

```
145     private void guardarTranscripcion(){
146         baseAutenticacion= FirebaseAuth.getInstance();
147
148         DbRef= FirebaseDatabase.getInstance().getReference().child("TRANSCRIPCIONES");
149         DateFormat df = new SimpleDateFormat( pattern: "EEEE, d MMMM yyyy, HH:mm:ss");
150         String date = df.format(Calendar.getInstance().getTime());
151         final Map<String,Object> mapUsuario=new HashMap<>();
152         mapUsuario.put( k: "titulo",tituloString);
153         mapUsuario.put( k: "fecha",date);
154         mapUsuario.put( k: "transcripcion",captura);
155         final String id2=baseAutenticacion.getCurrentUser().getUid();
156         DbRef.child(id2).child(tituloString).addValueEventListener(new ValueEventListener() {
157             @Override
158             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
159                 if(dataSnapshot.exists()){
160                     Toast.makeText(getActivity(), text: "El titulo ya existe, " +
161                             "CÁMBIALO, por favor", Toast.LENGTH_LONG).show();
162
163                 }else{
164                     DbRef.child(id2).child(tituloString).setValue(mapUsuario);
165                     Toast.makeText(getActivity(), text: "SU TRANSCRIPCIÓN SE GUARDO " +
166                             "CORRECTAMENTE", Toast.LENGTH_LONG).show();
167                     captura="";
168                     txtTranscripcion.setText(captura);
169                     tituloString="";
170                     tituloET.setText("");
171                     DbRef=null;
```

Código 2.76. Función guardarTranscripcion ()

2.5.27. VISUALIZAR PORTAFOLIO (PORTAFOLIOFRAGMENT)

Es la actividad donde se visualiza todas las transcripciones guardadas del usuario con la opción de seleccionar una para realizar cambios o eliminarla.

A continuación, se detalla líneas de código para el correcto funcionamiento de este Fragment como se observa en el código 2.77.

- **Código para creación de atributos**

```
27          private String mParam1;
28          private String mParam2;
29          RecyclerView recview;
30          myadapter adapter;
31          String idPropio;
32
33          private DatabaseReference DbRef;
34          private FirebaseAuth baseAutenticacion;
```

Código 2.77. Creación de atributos en PortafolioFragment

- **Código para funcionamiento**

De la línea 65 a 71 se apunta a un nodo de la base de datos y se inicializa los atributos. De la línea 73 a 79 se llama a la base de datos y gracias al adaptador se captura las transcripciones para ser mostradas visualmente como se puede ver en el código 2.78.

```java
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

    baseAutenticacion= FirebaseAuth.getInstance();
    final String id=baseAutenticacion.getCurrentUser().getUid();
    DbRef= FirebaseDatabase.getInstance().getReference().child("TRANSCRIPCIONES");

    View view= inflater.inflate(R.layout.fragment_portafolio, container, attachToRoot: false);
    recview=(RecyclerView)view.findViewById(R.id.recview);
    recview.setLayoutManager(new LinearLayoutManager(getContext()));

    FirebaseRecyclerOptions<model> options =
            new FirebaseRecyclerOptions.Builder<model>()
                    .setQuery(DbRef.child(id), model.class)
                    .build();
    adapter=new myadapter(options);
    recview.setAdapter(adapter);
    return view;
}
```

Código 2.78. Funcionamiento de PortafolioFragment

2.5.28. ELIMINAR TRANSCRIPCIÓN (ELIMINACIONFRAGMENT)

Es la actividad donde se guardará, editará o eliminará una transcripción.

A continuación, se detalla líneas de código para el correcto funcionamiento de este Fragment.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, título, botones y atributos pertenecientes a la base de datos y autenticación como se indica en el código 2.79.

```java
public class EliminacionFragment extends Fragment {
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    private String mParam1;
    private String mParam2;
    String titulo,captura;
    TextView tituloTV;
    EditText transcripcionET;
    private DatabaseReference DbRef;
    private FirebaseAuth baseAutenticacion;
    private ImageButton btnGuardar, btnEliminar,btnRegresar;

    public EliminacionFragment() {
    }

    public EliminacionFragment(String titulo) { this.titulo=titulo; }
```

Código 2.79. Creación de atributos en EliminacionFragment

- **Código para funcionamiento**

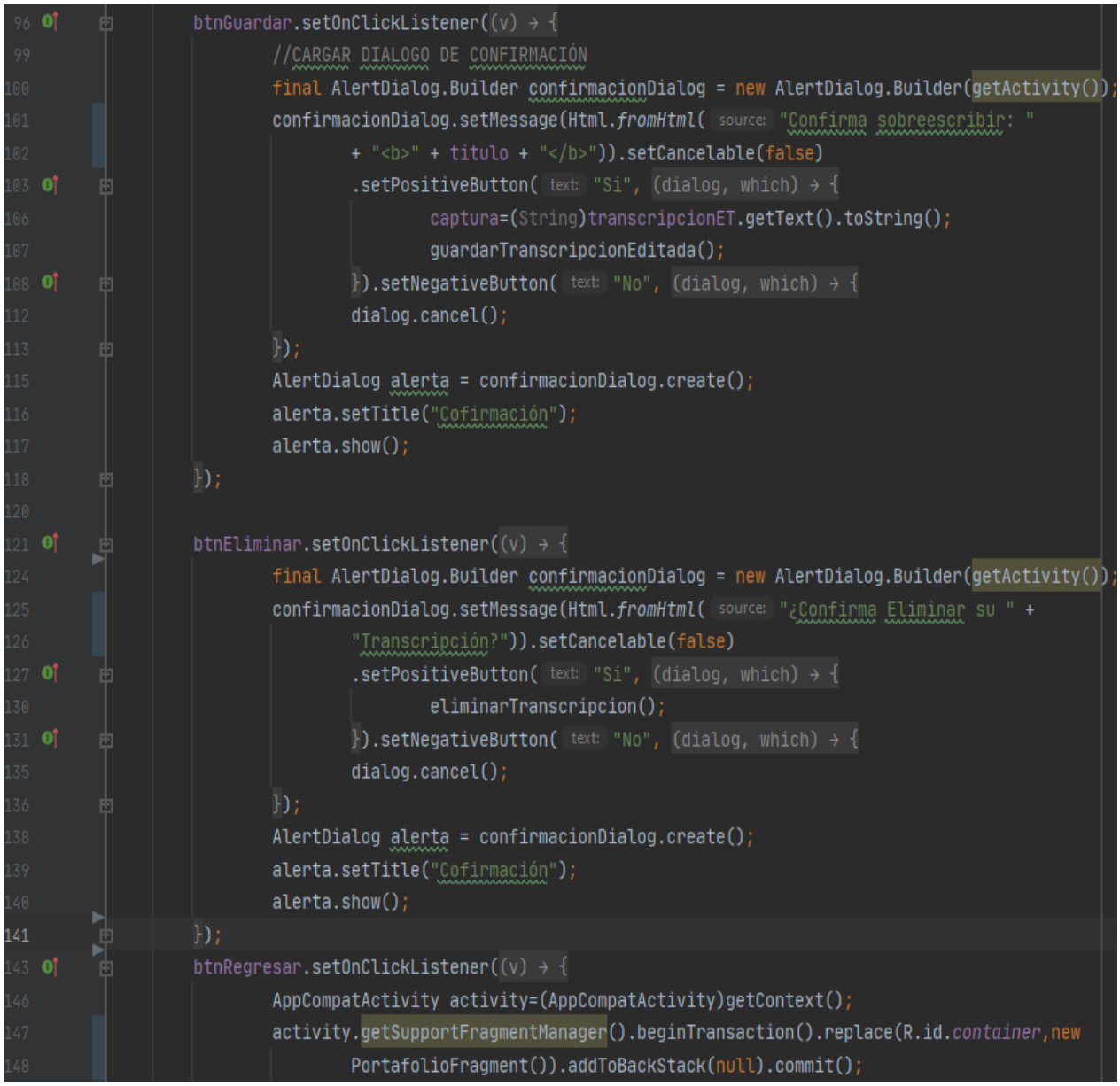
De la línea 74 a 82 se inicializa todos los atributos del fragment, de la línea 84 a 88 se captura el texto de la transcripción desde la base de datos como se observa en el código 2.80.

```
74          baseAutenticacion= FirebaseAuth.getInstance();
75          final String id=baseAutenticacion.getCurrentUser().getUid();
76          DbRef= FirebaseDatabase.getInstance().getReference( path: "TRANSCRIPCIONES");
77          btnGuardar=(ImageButton) view.findViewById(R.id.botonGuardar);
78          btnEliminar=(ImageButton) view.findViewById(R.id.botonEliminar);
79          btnRegresar=(ImageButton) view.findViewById(R.id.botonRegresar);
80          tituloTV=(TextView)view.findViewById(R.id.tituloTextoEliminacion);
81          transcripcionET=(EditText)view.findViewById((R.id.textoEntradaEliminacion));
82          tituloTV.setText(titulo);
83          //addListenerForSingleValueEvent
84          DbRef.child(id).child(titulo).addListenerForSingleValueEvent(new ValueEventListener() {
85              @Override
86              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
87                  if(dataSnapshot.exists()){
88                      transcripcionET.setText(dataSnapshot.child("transcripcion").getValue().toString());
89                  }
90              }
```

Código 2.80. Funcionamiento de EliminacionFragment

- **Código para funcionamiento de botones**

De la línea 97 a 118 se configura el funcionamiento de presionar el botón guardar en el cual se muestra un diálogo de alerta y al aceptar este diálogo se envía a la función *guardarTranscripcionEditada ()*, de la línea 120 a 141 se configura el funcionamiento de presionar el botón guardar en el cual se muestra un diálogo de alerta y al aceptar este diálogo se envía a la función *eliminarTranscripcion ()* y de la línea 144 a 150 se regresa a *PortafolioFragment ()* como se puede ver en el código 2.81.



```
96  btnGuardar.setOnClickListener((v) → {
99          //CARGAR DIALOGO DE CONFIRMACIÓN
100         final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder(getActivity());
101         confirmacionDialog.setMessage(Html.fromHtml( source: "Confirma sobreescribir: "
102                 + "<b>" + titulo + "</b>")).setCancelable(false)
103                 .setPositiveButton( text: "Si", (dialog, which) → {
106                         captura=(String)transcripcionET.getText().toString();
107                         guardarTranscripcionEditada();
108                 }).setNegativeButton( text: "No", (dialog, which) → {
112                 dialog.cancel();
113         });
115         AlertDialog alerta = confirmacionDialog.create();
116         alerta.setTitle("Cofirmación");
117         alerta.show();
118 });
120
121 btnEliminar.setOnClickListener((v) → {
124         final AlertDialog.Builder confirmacionDialog = new AlertDialog.Builder(getActivity());
125         confirmacionDialog.setMessage(Html.fromHtml( source: "¿Confirma Eliminar su " +
126                 "Transcripción?")).setCancelable(false)
127                 .setPositiveButton( text: "Si", (dialog, which) → {
130                         eliminarTranscripcion();
131                 }).setNegativeButton( text: "No", (dialog, which) → {
135                 dialog.cancel();
136         });
138         AlertDialog alerta = confirmacionDialog.create();
139         alerta.setTitle("Cofirmación");
140         alerta.show();
141 });
143 btnRegresar.setOnClickListener((v) → {
146         AppCompatActivity activity=(AppCompatActivity)getContext();
147         activity.getSupportFragmentManager().beginTransaction().replace(R.id.container,new
148                 PortafolioFragment()).addToBackStack(null).commit();
```

Código 2.81. Funcionamiento de botones en EliminacionFragment

- **Código para función eliminarTranscripcion () y *guardarTranscripcionEditada ()***

De la línea 155 a 163 se eliminará la transcripción buscándola en la base de datos. De la línea 165 a 173 se guardará los cambios realizados en la transcripción seleccionada como se observa en el código 2.82.

```
155     public void eliminarTranscripcion(){
156         final String id=baseAutenticacion.getCurrentUser().getUid();
157         //DbRef.child(id).child(titulo).setValue(null);
158         DbRef.child(id).child(titulo).removeValue();
159         DbRef=null;
160         Toast.makeText(getActivity(),  text: "TRANSCRIPCION ELIMINADA", Toast.LENGTH_SHORT).show();
161         //getFragmentManager().beginTransaction().remove(this).commit();
162         AppCompatActivity activity=(AppCompatActivity)getContext();
163         activity.getSupportFragmentManager().beginTransaction().replace(R.id.container,
164                 new PortafolioFragment()).addToBackStack(null).commit();
165     }
166     //guardar
167     public void guardarTranscripcionEditada(){
168         final String id=baseAutenticacion.getCurrentUser().getUid();
169         DbRef.child(id).child(titulo).child("transcripcion").setValue(captura);
170         DbRef=null;
171         Toast.makeText(getActivity(),  text: "SU TRANSCRIPCIÓN SE GUARDO CORRECTAMENTE",
172                 Toast.LENGTH_LONG).show();
173         //getFragmentManager().beginTransaction().remove(this).commit();
174         AppCompatActivity activity2=(AppCompatActivity)getContext();
175         activity2.getSupportFragmentManager().beginTransaction().replace(R.id.container,
176                 new PortafolioFragment()).addToBackStack(null).commit();
```

Código 2.82. Funcionamiento de eliminar y guardar transcripción

2.5.29. VISUALIZAR UBICACIÓN (MAPAACTIVITY)

Es la actividad donde se visualiza la ubicación actual del usuario y la opción de enviar esta ubicación a un contacto de WhatsApp.

A continuación, se detalla líneas de código para el correcto funcionamiento de esta Activity.

- **Código para creación de atributos**

Se tiene los diferentes atributos que se utilizan para la inicialización de elementos, cargar la ubicación actual en el mapa y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.83.

```
44  public class MapaActivity extends FragmentActivity implements OnMapReadyCallback {
45
46      private GoogleMap mMap;
47      private Button btnEnviar;
48      private String ubicacion,latitud,longitud;
49      private String contador="0";
50      private DatabaseReference DbRef;
51      private FirebaseAuth baseAutenticacion;
52
```

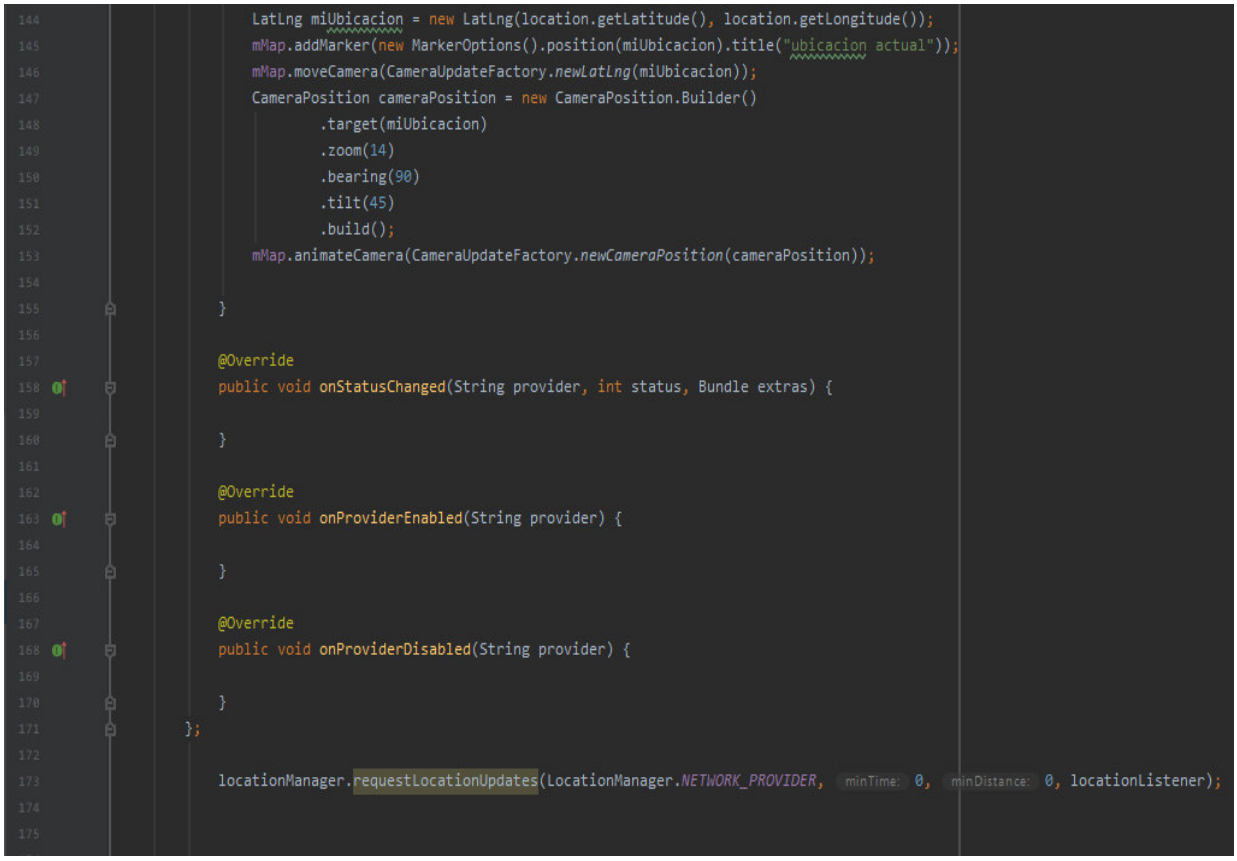
Código 2.83. Creación de atributos en MapaActivity

- **Código para función *onMapReady ()***

Esta función permitirá en la línea 105 comprobar permisos para utilizar ubicación actual, de la línea 108 a 113 se escuchará la ubicación actual dentro de la aplicación, de la línea 116 a 118 se creará un string con formato de Google Maps para más tarde enviar a WhatsApp, de la línea 119 a 134 se creará un mapa de atributos con fecha, latitud y longitud y se lo guardará en la base de datos. De la línea 144 a 153 se visualizará el punto de ubicación actual en el mapa. En la línea 173 se tiene la acción de escuchar el punto de ubicación actual como se puede ver en el código 2.84 y 2.85.

```
103  public void onMapReady(GoogleMap googleMap) {
104      mMap = googleMap;
105      if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
106              ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
107          return;
108      }
109      mMap.setMyLocationEnabled(true);
110
111      mMap.getUiSettings().setMyLocationButtonEnabled(false);
112
113      LocationManager locationManager = (LocationManager) MapaActivity.this.getSystemService(Context.LOCATION_SERVICE);
114      LocationListener locationListener = new LocationListener() {
115          @Override
116          public void onLocationChanged(Location location) {
117              ubicacion=(String)"https://www.google.com/maps/place/"+location.getLatitude()+","+location.getLongitude();
118              latitud=String.valueOf(location.getLatitude());
119              longitud=String.valueOf(location.getLongitude());
120              if(contador.equals("0")){
121                  final String id=baseAutenticacion.getCurrentUser().getUid();
122                  DbRef.child("USUARIOS").child("PRINCIPAL").child(id).addListenerForSingleValueEvent(new ValueEventListener() {
123                      @Override
124                      public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
125                          if(dataSnapshot.exists()){}
126                          DateFormat df = new SimpleDateFormat( pattern: "EEEE, d MMMM yyyy, HH:mm:ss");
127                          String date = df.format(Calendar.getInstance().getTime());
128                          final Map<String,Object> mapUsuario=new HashMap<>();
129                          mapUsuario.put( k: "fecha",date);
130                          mapUsuario.put( k: "latitud",latitud);
131                          mapUsuario.put( k: "longitud",longitud);
132                          DbRef.child("UBICACION").child(id).push().setValue(mapUsuario);
133                          DbRef=null;
134                          contador="1";
135
136                      }
```

Código 2.84. Función onMapReady () en MapaActivity



```
144                 LatLng miUbicacion = new LatLng(location.getLatitude(), location.getLongitude());
145                 mMap.addMarker(new MarkerOptions().position(miUbicacion).title("ubicacion actual"));
146                 mMap.moveCamera(CameraUpdateFactory.newLatLng(miUbicacion));
147                 CameraPosition cameraPosition = new CameraPosition.Builder()
148                         .target(miUbicacion)
149                         .zoom(14)
150                         .bearing(90)
151                         .tilt(45)
152                         .build();
153                 mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
154
155             }
156
157             @Override
158             public void onStatusChanged(String provider, int status, Bundle extras) {
159
160             }
161
162             @Override
163             public void onProviderEnabled(String provider) {
164
165             }
166
167             @Override
168             public void onProviderDisabled(String provider) {
169
170             }
171         };
172
173             locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,  minTime: 0,  minDistance: 0, locationListener);
174
175
```

Código 2.85. Continuación de función onMapReady () en MapaActivity

- **Código para enviar ubicación a WhatsApp**

De la línea 60 a 70 se comprueba una conexión básica y activa con GooglePlayServices y se inicializa el objeto *map*. De la línea 74 a 87 se utiliza in *intent* para comprobar que está instalado WhatsApp en el dispositivo y si es el caso se puede enviar la ubicación a un contacto como se observa en el código 2.86.

```
        int status= GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
        if(status== ConnectionResult.SUCCESS) {


            SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                    .findFragmentById(R.id.map);
            mapFragment.getMapAsync( onMapReadyCallback: this);
        }else {
            Dialog dialog=GooglePlayServicesUtil.getErrorDialog(status,(Activity) getApplicationContext(), i1: 10);
            dialog.show();
        }

        btnEnviar=(Button)findViewById(R.id.buttonUbicacion);
        ubicacion=null;
        btnEnviar.setOnClickListener((v) -> {
                Intent intent = new Intent(Intent.ACTION_SEND);
                intent.setType("text/plain");
                intent.setPackage("com.whatsapp");
                intent.putExtra(Intent.EXTRA_TEXT, ubicacion);
                try{
                    startActivity(intent);
                }catch (android.content.ActivityNotFoundException ex){
                    Toast.makeText( context: MapaActivity.this, text: "no tiene instalado Whatsapp", Toast.LENGTH_SHORT).show();
                }
                Toast.makeText( context: MapaActivity.this, text: "Presione su ubicación en el mapa", Toast.LENGTH_LONG).show();
        });
        getLocalizacion();
    }

    private void getLocalizacion() {
        int permiso = ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION);
        if(permiso == PackageManager.PERMISSION_DENIED){
            if(ActivityCompat.shouldShowRequestPermissionRationale( activity: this, Manifest.permission.ACCESS_FINE_LOCATION)){
            }else{
                ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 1);
            }
        }
```

Código 2.86. Enviar ubicación a WhatsApp en MapaActivity

2.5.30. INFORMACIÓN DE TRANSCRPCIONES (REPORTEFRAGMENT)

Es el fragment donde se visualiza la información básica de todas las transcripciones del usuario y se puede descargar un PDF con la información anterior.

A continuación, se detalla líneas de código para el correcto funcionamiento de ese Fragment.

- **Código para creación de atributos**

Se crea botón para crear el PDF, adaptador, constructor del fragment para recibir el ID del usuario referente a la base de datos y atributos pertenecientes a la base de datos y autenticación como se observa en el código 2.87.

```
24  public class ReporteFragment extends Fragment {
25
26      private static final String ARG_PARAM1 = "param1";
27      private static final String ARG_PARAM2 = "param2";
28
29
30      private String mParam1;
31      private String mParam2;
32      RecyclerView recview;
33      myadapterreporte adapter;
34      String idPropio;
35      Button PDF;
```

Código 2.87. Creación de atributos en ReporteFragment

- **Código para funcionamiento**

De la línea 69 a 76 se inicializa todos los atributos, de la línea 77 a 81 se tiene la acción cuando se pulsa el botón para obtener el archivo PDF llamando a *Prueba ();* de la línea 84 a 90 se llama a la base de datos para obtener todas las transcripciones y mostrarlas en pantalla como se indica en el código 2.88.

```
66  @   public View onCreateView(LayoutInflater inflater, ViewGroup container,
67                               Bundle savedInstanceState) {
68
69          baseAutenticacion= FirebaseAuth.getInstance();
70          final String id=baseAutenticacion.getCurrentUser().getUid();
71          DbRef= FirebaseDatabase.getInstance().getReference().child("TRANSCRIPCIONES");
72
73          View view= inflater.inflate(R.layout.fragment_reporte, container,  attachToRoot: false);
74          recview=(RecyclerView)view.findViewById(R.id.recviewReporte);
75          recview.setLayoutManager(new LinearLayoutManager(getContext()));
76          PDF=(Button)view.findViewById(R.id.buttonReporte);
77          PDF.setOnClickListener((v) -> {
80
81                  startActivity(new Intent(getActivity(), Prueba.class));
82          });
84                  FirebaseRecyclerOptions<modelReporte> options =
85                  new FirebaseRecyclerOptions.Builder<modelReporte>()
86                          .setQuery(DbRef.child(id), modelReporte.class)
87                          .build();
88          adapter=new myadapterreporte(options);
89          recview.setAdapter(adapter);
90          return view;
91      }
```

Código 2.88. Funcionamiento de ReporteFragment

2.5.31. PRUEBA

Es la actividad donde se crea la estructura del PDF, se obtiene datos de la base de datos y se crea un tiempo de espera para evitar errores ya que la base de datos funciona en tiempo real. Tiene el mismo funcionamiento indicado en PRUEBAFAMILIAR.

2.5.32. CHATFRAGMENT

Es la actividad donde se crea la estructura para recibir en enviar mensajes ente el usuario Familiar y su usuario Principal con quien tiene sincronismo. Tiene el mismo funcionamiento indicado en CHATFRAGMENTFAMILIAR.

3. RESULTADOS Y DISCUSIÓN

Se generó varias pruebas para comprobar el correcto funcionamiento del aplicativo, tanto para cada módulo como para la totalidad.

Entre las pruebas realizada están:

- Pruebas de creación de usuarios.
- Pruebas de conectividad entre la aplicación y la base de datos.
- Pruebas de autenticación de usuarios.
- Pruebas de transcripción.
- Pruebas de guardado de transcripción.
- Pruebas de edición y eliminado de transcripciones.
- Prueba de mensajes.
- Prueba de ubicación.
- Prueba de PDFs.
- Prueba de informes generales.
- Prueba recuperación de clave.

3.1. PRUEBAS DE FUNCIONAMIENTO

A continuación, se indican ejemplos de pruebas concluidas.

3.1.1. PRUEBAS DE CREACIÓN DE USUARIOS

Para esta prueba se creó tres tipos de usuarios cada uno perteneciente al usuario Principal, Familiar y Administrador. Se selecciona el tipo de rol, un nombre, un correo y una clave. El momento de crear el usuario se enviará un mensaje al correo ingresado para confirmar que el correo existe. El usuario Administrador lo puede crear solo un usuario Administrador ya existente como se observa en las Figuras 3.1, 3.2 y 3.3.

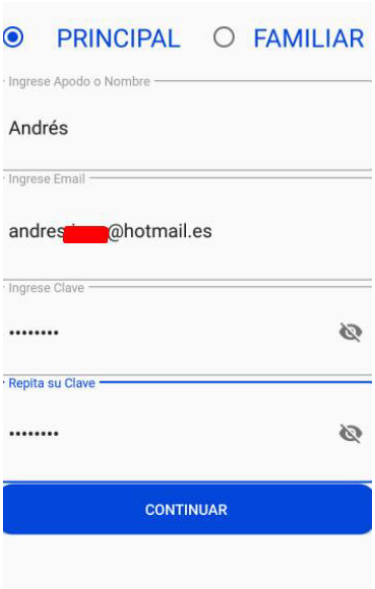


Figura 3.1. Creación de usuario

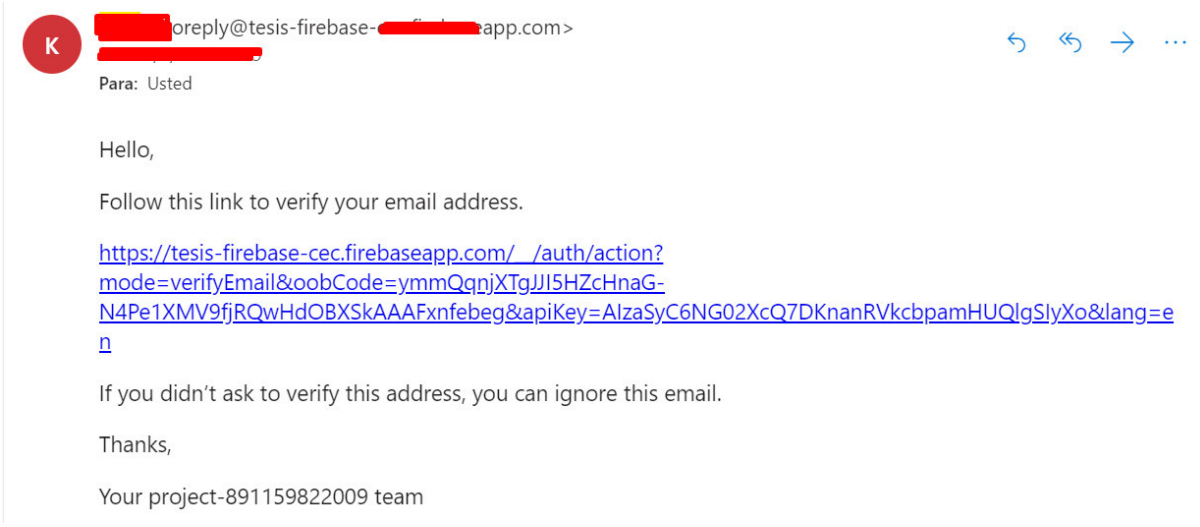


Figura 3.2 Confirmación en correo

Figura 3.3. Registro en autenticación de firebase

3.1.2. PRUEBAS DE CONECTIVIDAD

Para la prueba de conectividad se debe asegurar que al momento de crear el usuario todos los datos pertenecientes a este, se generen en Realtime Database de Firebase, en la figura se puede observar el nodo USUARIOS y los nodos hijos ADMINISTRADOR, FAMILIAR y PRINCIPAL. Se generó nodos hijos independientes para optimizar la velocidad de la aplicación con la base de datos como se observa en la Figura 3.4.

Figura 3.4. Registro en base de datos de firebase

3.1.3. AUTENTICACIÓN DE USUARIOS

En la interfaz inicial del prototipo se muestra dos espacios donde se ingresará el correo y la clave, al presionar el botón “INGRESAR” se hará un llamado a la base de datos para confirmar que el usuario existe y la clave es correcta lo que permitirá ingresar al menú correspondiente como se observa en la Figura 3.5. En las Figura 3.6 se muestra el menú desplegable para el usuario principal, En las Figura 3.7 se muestra el menú desplegable para el usuario Familiar y En las Figura 3.8 se muestra el menú desplegable para el usuario Administrador.

Figura 3.5. Inicio de sesión

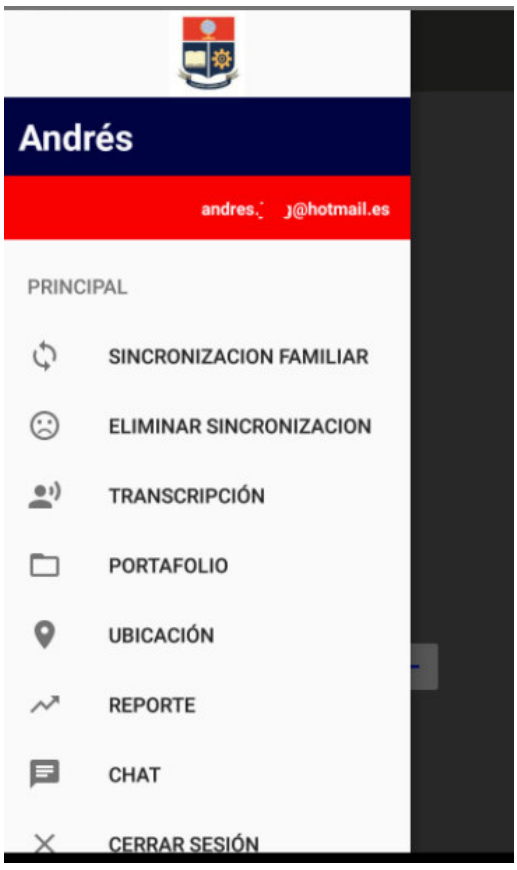


Figura 3.6. Menú usuario Principal

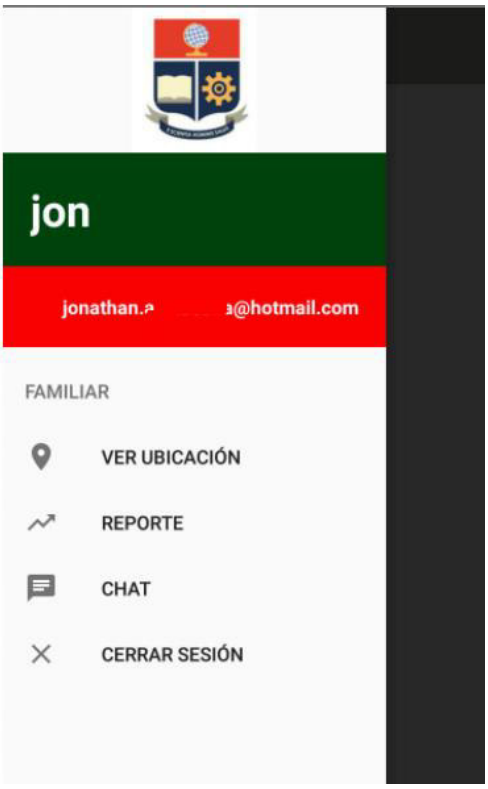


Figura 3.7. Menú usuario Familiar

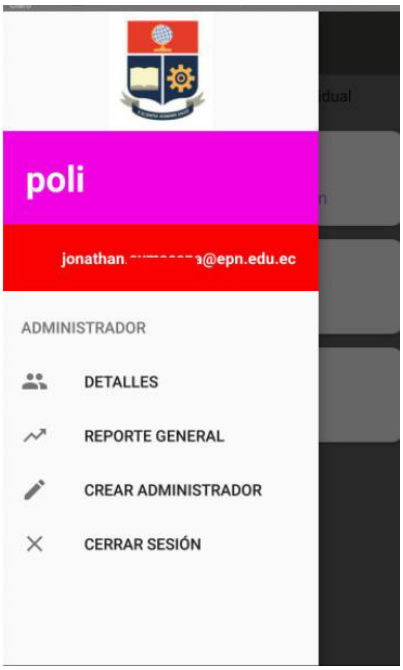


Figura 3.8. Menú usuario Administrador

3.1.4. PRUEBAS DE TRANSCRIPCIÓN

Al momento de seleccionar el botón “Escuchar” se llamará al servicio más importante de la aplicación el cual permitirá escuchar la voz para poder transcribirla a texto, se tiene la opción de pausar la transcripción o iniciar nuevamente. A medida que la voz fluya se puede visualizar en la pantalla y se guardará en un buffer el cual se presentará al finalizar la transcripción como se indica en las Figuras 3.9, 3.10 y 3.11.

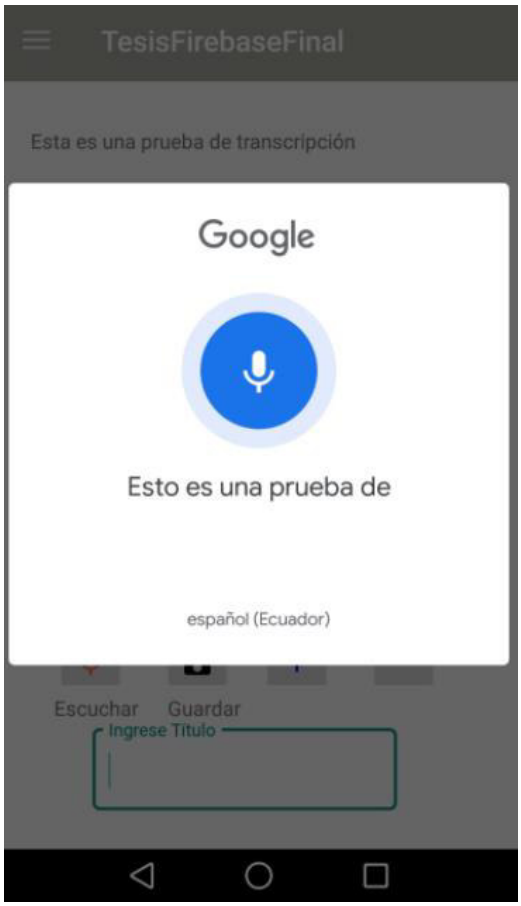


Figura 3.9. Transcripción en español

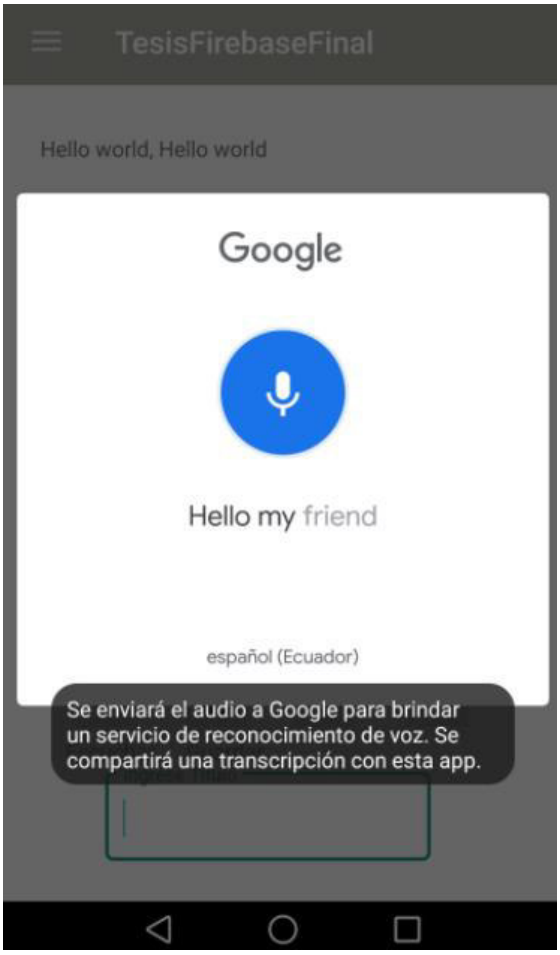


Figura 3.10. Transcripción en ingles

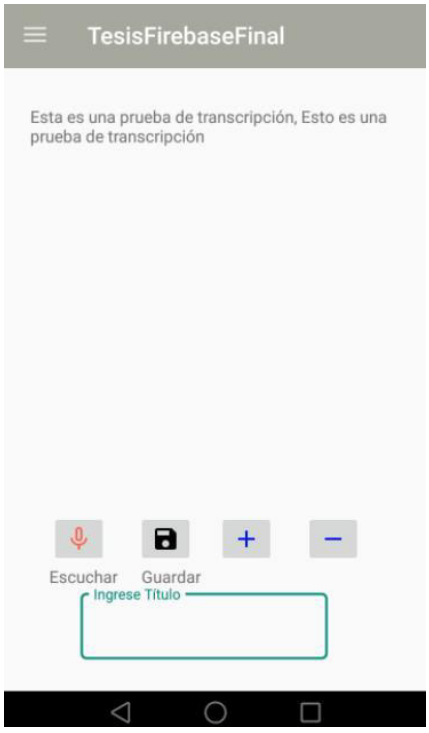


Figura 3.11. Resultado final de transcripción

3.1.5. PRUEBAS DE GUARDAR TRANSCRIPCIÓN

En este punto se selecciona el botón “Guardar” el cual pedirá confirmar el título de la transcripción a guardar para más tarde guardarla en un nuevo nodo independiente dedicado a transcripciones; esto con el fin de optimizar velocidades en el prototipo como se puede ver en las Figuras 3.12 y 3.13.

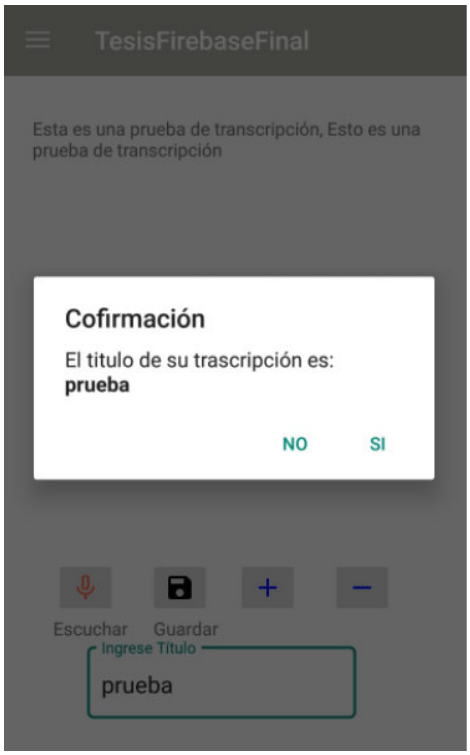


Figura 3.12. Guardar Transcripción

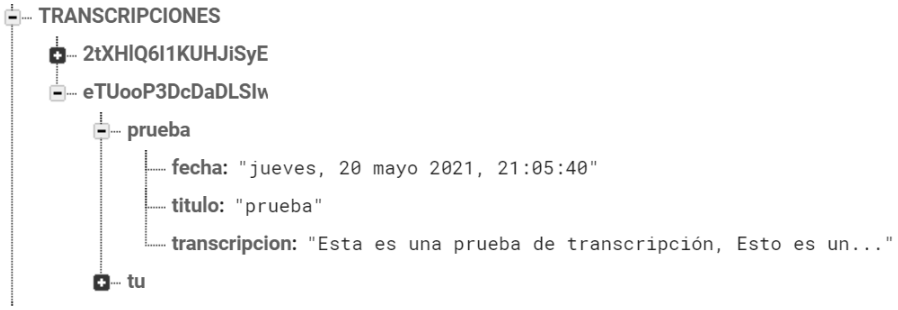


Figura 3.13. Transcripción guardada en base de datos

3.1.6. PRUEBAS DE EDICIÓN Y ELIMINADO

En esta sección se selecciona “PORTAFOLIO” del menú desplegable para después seleccionar la transcripción a editar, una vez realizado los cambios se puede guardar o

eliminar para luego modificar en la base de datos como se indica en las Figuras 3.14, 3.15, 3.16 y 3.17.

Figura 3.14. Transcripción original

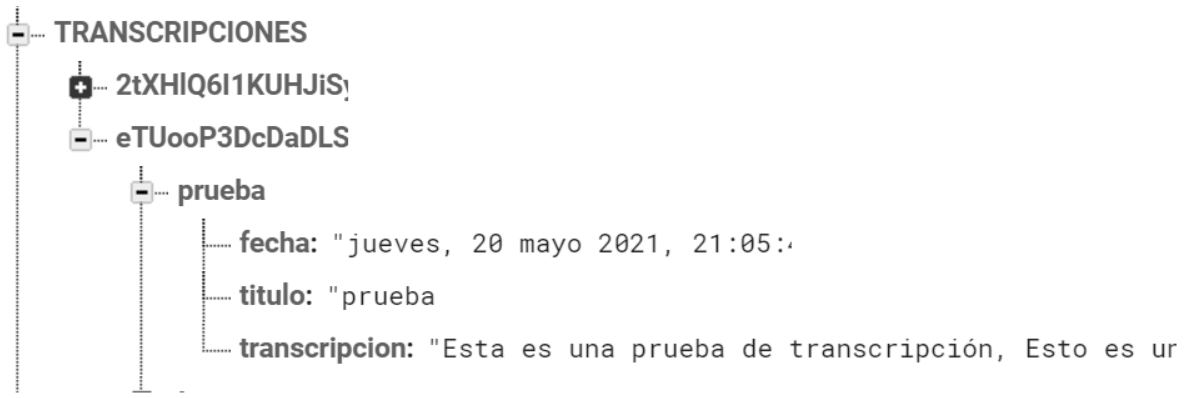


Figura 3.15. Transcripción original en base de datos

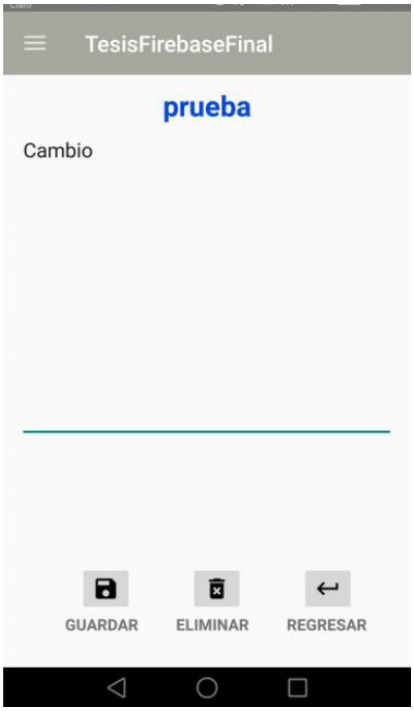


Figura 3.16. Cambio o edición a transcripción

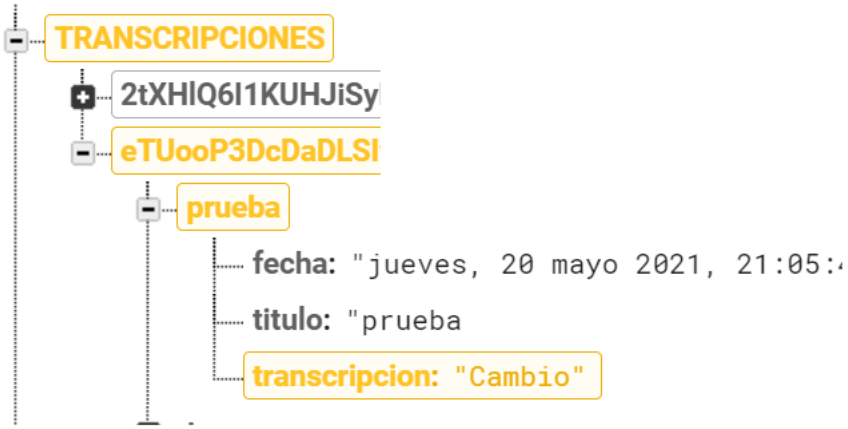


Figura 3.17. Cambio en base de datos

3.1.7. PRUEBA DE MENSAJES

En este módulo se podrá enviar mensajes entre el usuario Principal y su Familiar sincronizado, se utiliza un nuevo nodo en la base de datos donde se almacenará todos los mensajes en el usuario correspondiente; cada mensaje tendrá un “estado” y el “texto” como se observa en las Figuras 3.18 y 3.19.

Figura 3.18. Mensaje enviado y recibido

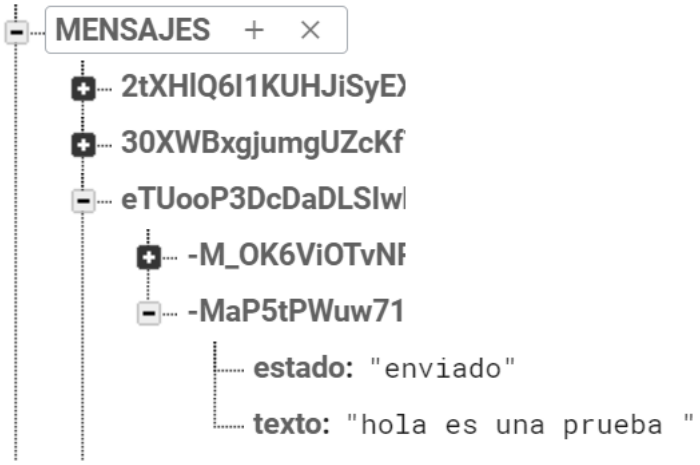


Figura 3.20. Mensaje guardado en base de datos

3.1.8. PRUEBA DE UBICACIÓN

En este módulo se puede ver la ubicación actual en el usuario Principal y se puede enviar esta a un contacto de WhatsApp. Al momento de ingresar al mapa, el punto de ubicación se guardará en la base de datos por fecha, longitud y latitud para más tarde capturar estos datos y mostrarlos al usuario Familiar con quien se tiene sincronismo como se puede ver en las Figuras 3.21, 3.22, 3.23 y 3.24.

Figura 3.21 Ubicación usuario Principal

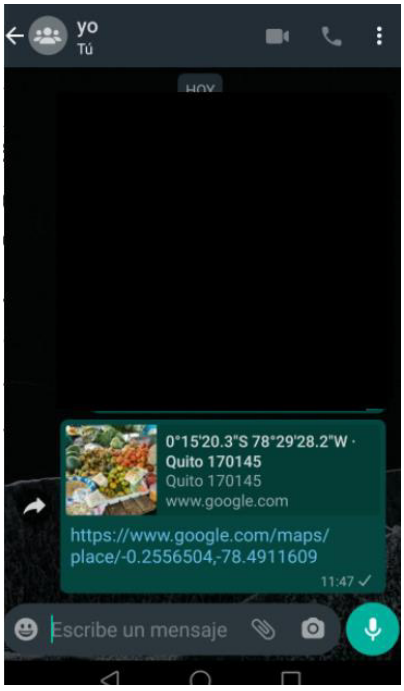


Figura 3.22. Enviar ubicación por WhatsApp

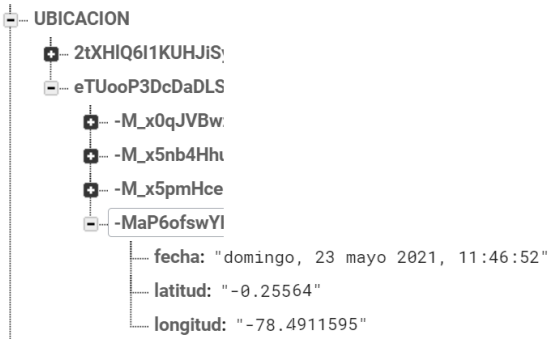


Figura 3.23. Ubicación guardada en base de datos

Figura 3.24. Ver ubicación desde usuario Familiar

3.1.9. PRUEBA DE GENERACIÓN DE PDF

Para esta prueba se captura todos los títulos y fechas de las transcripciones de un usuario Principal desde la base de datos, se las presente en la pantalla del dispositivo y se puede guardar el PDF. Para visualizar el PDF en la mayoría de casos es necesario instalar una aplicación para ver carpetas del dispositivo como puede ser Cx Explorador de Archivos. El formato del PDF es simple y entendible como se indica en la Figura 3.25.

Figura 3.25. PDF

3.1.10. PRUEBA DE INFORMES GENERALES

En esta prueba se realiza una búsqueda en la base de datos ya sea individual para obtener información de cada usuario o general para obtener información global de la base de datos como se observa en la Figura 3.26 y 3.27.

Figura 3.26. Informe individual

Figura 3.27. Informe global

3.1.11. PRUEBA RECUPERACIÓN DE CLAVE

Para esta prueba se debe seleccionar el botón “RECUPERAR CONTRASEÑA” de la pantalla de inicio; luego ingresar el correo, se enviará un correo en donde se puede ingresar la nueva clave como se observa en las Figuras 3.28, 3.29 y 3.30.

Figura 3.28. Ingresar el correo a recuperar contraseña

Figura 3.29. Correo de cambio de contraseña

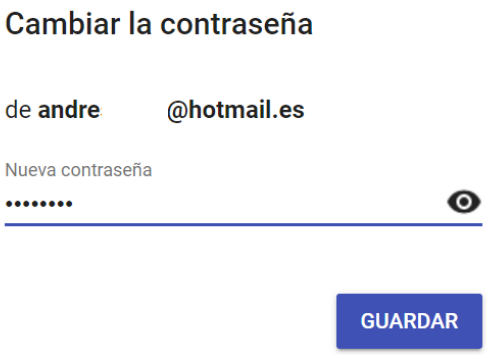


Figura 3.30. Ingreso de nueva contraseña

3.2. ENCUESTAS DE VALIDACIÓN

Se solicitó a 10 personas entre adultos mayores, adultos y jóvenes que en algunos casos tienen problemas auditivos, la realización de las pruebas referente a la aplicación. Hay que tener en cuenta que existen tres tipos de roles por lo que se tiene la evaluación de 2 personas para el rol Administrador, 4 personas para el rol Principal y 4 personas para al rol Familiar. Después de que cada usuario utilizó la aplicación, se pidió llenar una encuesta de validación, los formatos de las encuestas llenas se encuentran en anexos. [25]

3.2.1. ROL ADMINISTRADOR

La primera pregunta busca comprobar si es posible crear el tipo de usuario. Los resultados indican que el 100% de los encuestados pueden crear el tipo de usuario como se indica en la Figura 3.31.

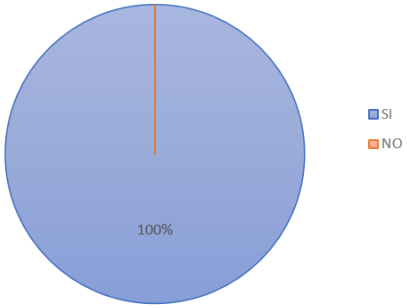


Figura 3.31. Respuesta a la primera pregunta a Administrador

La segunda pregunta busca comprobar si al momento de crear su usuario pudo confirmarlo a través de un mensaje a su correo personal. Los resultados indican que el 100% de los encuestados reciben el correo para confirmarlo como se observa en la Figura 3.32.

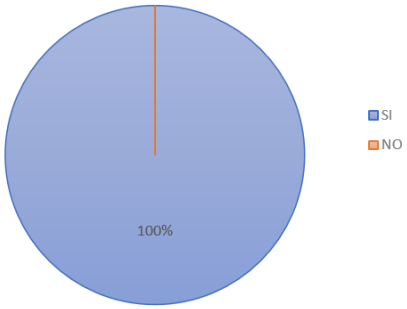


Figura 3.32. Respuesta a la segunda pregunta a Administrador

La tercera pregunta busca comprobar si es posible entrar al menú de su rol. Los resultados indican que el 100% de los encuestados pueden ingresar al menú como se puede ver en la Figura 3.33.

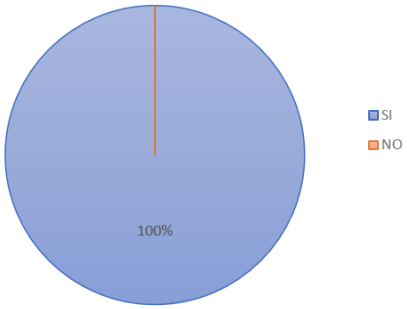


Figura 3.33. Respuesta a la tercera pregunta a Administrador

La cuarta pregunta busca comprobar si al momento de cerrar su aplicación y volver a ingresar se le solicitó nuevamente la clave. Los resultados indican que el 100% de los encuestados no ingresan nuevamente la clave, sino que ingresa directamente a la interfaz de su rol como se indica en la Figura 3.34.

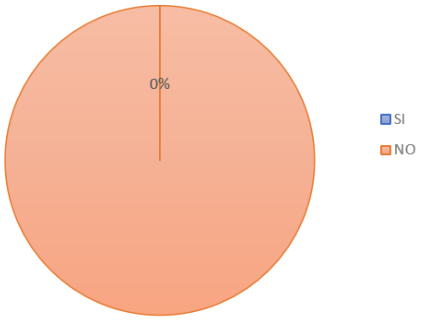


Figura 3.34. Respuesta a la cuarta pregunta a Administrador

La quinta pregunta busca comprobar si es posible ver un detalle general de cada usuario principal. Los resultados indican que el 100% de los encuestados pueden ver detalles generales de cada usuario Principal perteneciente a la base de datos como se observa en la Figura 3.35.

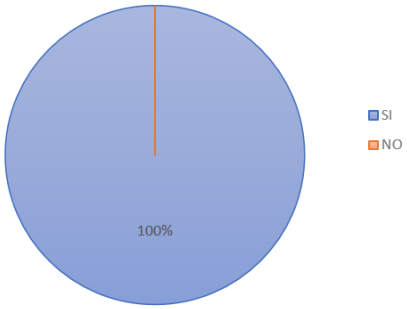


Figura 3.35. Respuesta a la quinta pregunta a Administrador

La sexta pregunta busca comprobar si es posible ver un detalle general de la aplicación. Los resultados indican que el 100% de los encuestados pueden ver detalles generales perteneciente a la base de datos como se puede ver en la Figura 3.36.

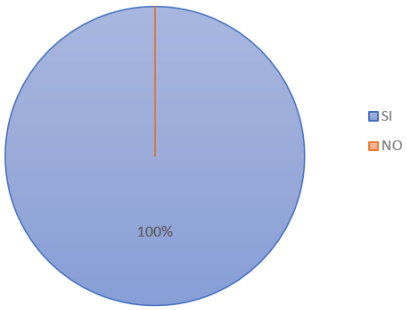


Figura 3.36. Respuesta a la sexta pregunta a Administrador

La séptima pregunta busca comprobar si es posible regresar a cualquier parte que desee utilizando el menú desplegable. Los resultados indican que el 100% de los encuestados pueden dirigirse a cualquier parte del menú desplegable como se observa en la Figura 3.37.

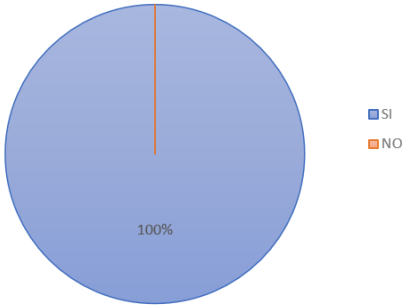


Figura 3.37. Respuesta a la séptima pregunta a Administrador

La octava pregunta busca comprobar si es posible cerrar sesión. Los resultados indican que el 100% de los encuestados pueden cerrar sesión como se indica en la Figura 3.38.

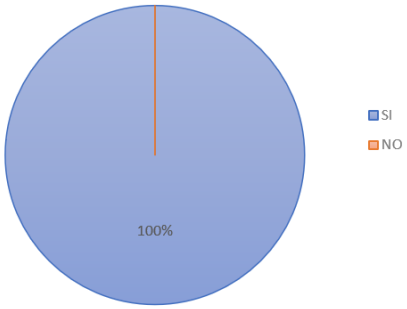


Figura 3.38. Respuesta a la octava pregunta a Administrador

La novena pregunta busca comprobar qué le pareció la idea de poder ver informes generales respetando la privacidad de cada usuario. Los resultados indican que el 50% de los encuestados toman esta pregunta como “BUENO” y 50% como “MUY BUENO” como se muestra en la Figura 3.39.

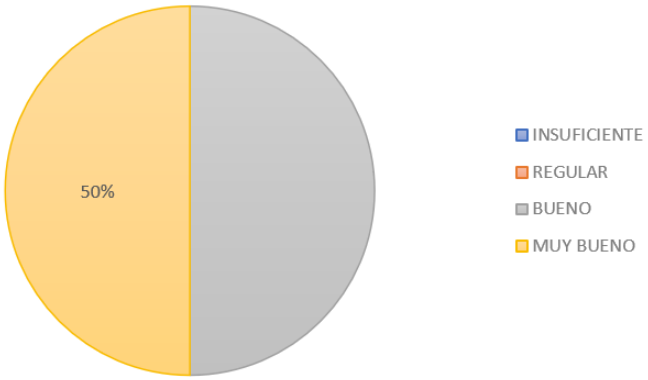


Figura 3.39. Respuesta a la novena pregunta a Administrador

La décima pregunta busca comprobar qué le pareció el manejo de la aplicación respecto a velocidad. Los resultados indican que el 100% de los encuestados toman esta pregunta como “MUY BUENO” como se observa en la Figura 3.40.

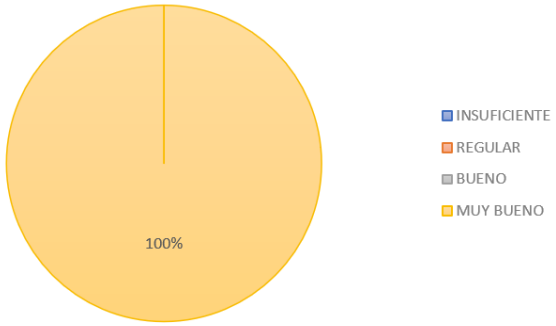


Figura 3.40. Respuesta a la décima pregunta a Administrador

La decimoprimera pregunta busca comprobar qué le pareció la parte gráfica (dibujos, letras, imágenes, colores). Los resultados indican que el 50% de los encuestados toman esta pregunta como “BUENO” y 50% como “MUY BUENO” como se puede ver en la Figura 3.41.

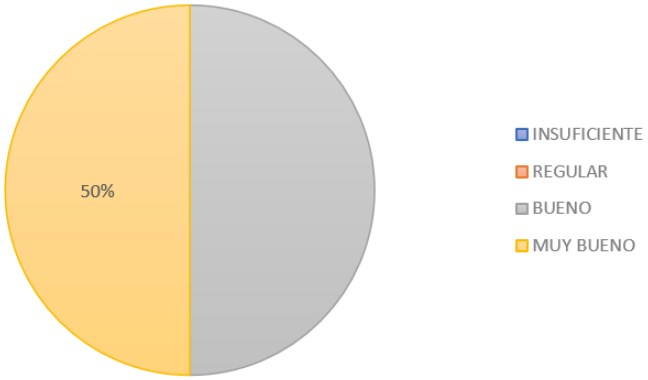


Figura 3.41. Respuesta a la decimoprimera pregunta

3.2.2. ROL FAMILIAR

La primera pregunta busca comprobar si es posible crear el tipo de usuario. Los resultados indican que el 100% de los encuestados pueden crear el tipo de usuario como se indica en la Figura 3.42.

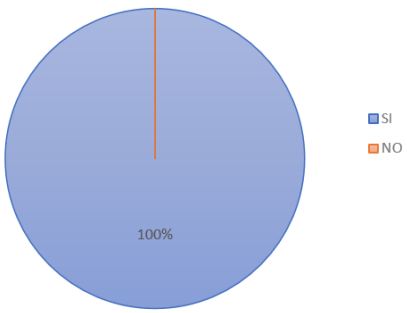


Figura 3.42. Respuesta a la primera pregunta a Familiar

La segunda pregunta busca comprobar si al momento de crear su usuario pudo confirmarlo a través de un mensaje a su correo personal. Los resultados indican que el 100% de los encuestados reciben el correo para confirmarlo como se muestra en la Figura 3.43.

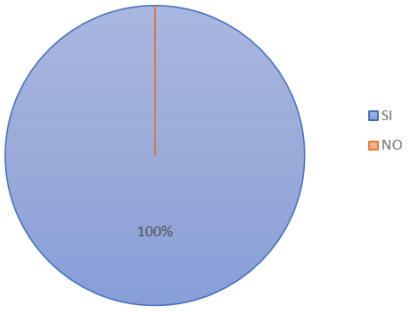


Figura 3.43. Respuesta a la segunda pregunta a Familiar

La tercera pregunta busca comprobar si es posible entrar al menú de su rol. Los resultados indican que el 100% de los encuestados pueden ingresar al menú como se observa en la Figura 3.44.

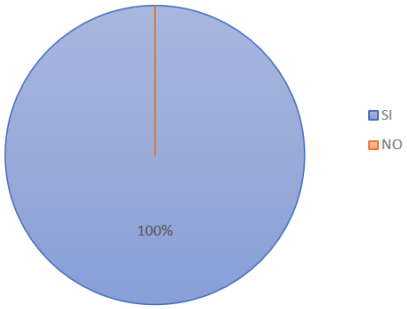


Figura 3.44. Respuesta a la tercera pregunta a Familiar

La cuarta pregunta busca comprobar si al momento de cerrar su aplicación y volver a ingresar se le solicitó nuevamente la clave. Los resultados indican que el 100% de los encuestados no ingresan nuevamente la clave, sino que ingresa directamente a la interfaz de su rol como se indica en la Figura 3.45.

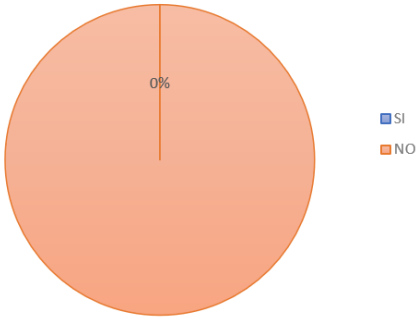


Figura 3.45. Respuesta a la cuarta pregunta a Familiar

La quinta pregunta busca comprobar si es posible ver los diferentes puntos o recorridos de su usuario principal en el mapa por fecha, día y hora sin utilizar ningún otro aplicativo como Google Maps. Los resultados indican que el 100% de los encuestados pueden ver los detalles como se puede ver en la Figura 3.46.

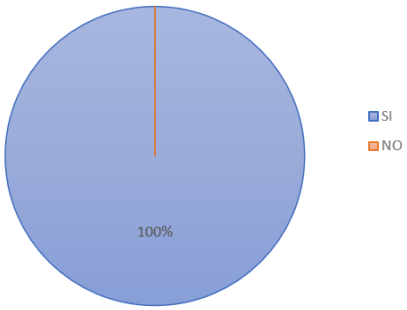


Figura 3.46. Respuesta a la quinta pregunta a Familiar

La sexta pregunta busca comprobar si es posible ver un reporte en la aplicación y también descargarlo en PDF. Los resultados indican que el 100% de los encuestados pueden generar el PDF como se observa en la Figura 3.47.

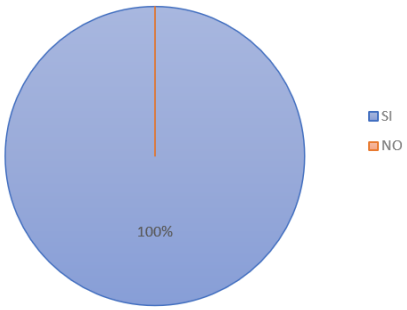


Figura 3.47. Respuesta a la sexta pregunta a Familiar

La séptima pregunta busca comprobar si es posible chatear con su usuario principal sincronizado. Los resultados indican que el 100% de los encuestados pueden chatear como se muestra en la Figura 3.48.

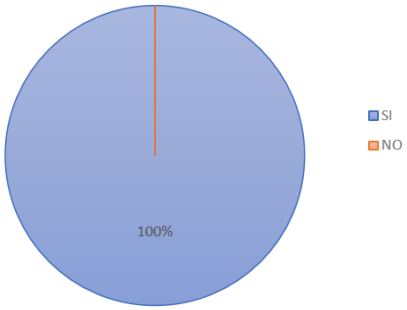


Figura 3.48. Respuesta a la séptima pregunta a Familiar

La octava pregunta busca comprobar si encontró algún tipo de confirmación al momento de querer salir de la aplicación. Los resultados indican que el 100% de los encuestados recibieron la confirmación como se observa en la Figura 3.49.

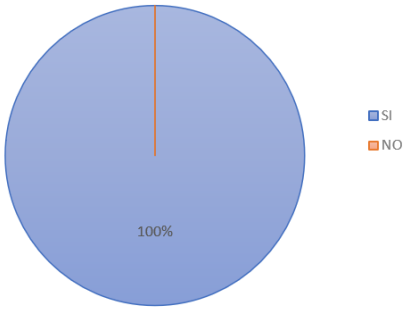


Figura 3.49. Respuesta a la octava pregunta a Familiar

La novena pregunta busca comprobar si es posible regresar a cualquier parte que desee utilizando el menú desplegable. Los resultados indican que el 100% de los encuestados pueden dirigirse a cualquier parte del menú desplegable como se indica en la Figura 3.50.

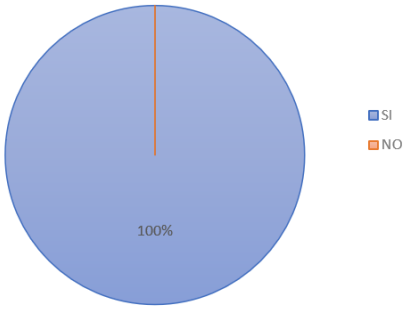


Figura 3.50. Respuesta a la novena pregunta a Familiar

La décima pregunta busca comprobar si es posible cerrar sesión. Los resultados indican que el 100% de los encuestados pueden cerrar sesión como se puede ver en la Figura 3.51.

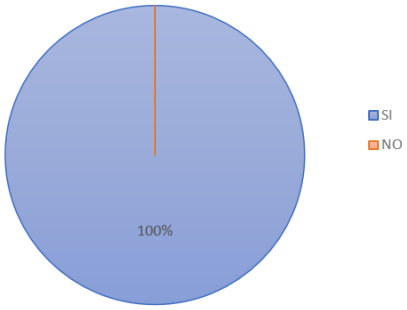


Figura 3.51. Respuesta a la décima pregunta a Familiar

La decimoprimera pregunta busca comprobar qué le pareció la idea de poder ver ubicaciones y enviar mensajes a un usuario con problemas auditivos teniendo en

cuenta que el usuario principal tendrá la autoría de elegir con quien compartir este privilegio. Los resultados indican que el 50% de los encuestados toman esta pregunta como “BUENO” y 50% como “MUY BUENO” como se muestra en la Figura 3.52.

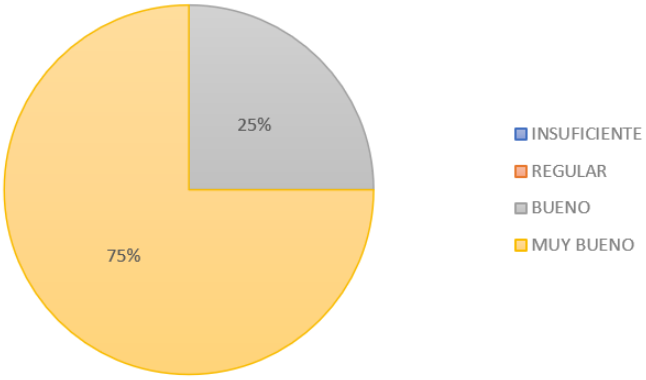


Figura 3.52. Respuesta a la decimoprimera pregunta a Familiar

La decimosegunda pregunta busca comprobar qué le pareció el manejo de la aplicación respecto a velocidad. Los resultados indican que el 100% de los encuestados toman esta pregunta como “MUY BUENO” como se observa en la Figura 3.53.

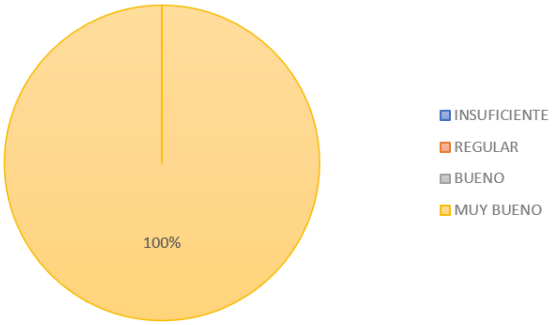


Figura 3.53. Respuesta a la decimosegunda pregunta a Familiar

La decimotercera pregunta busca comprobar qué le pareció la parte gráfica (dibujos, letras, imágenes, colores). Los resultados indican que el 75% de los encuestados toman esta pregunta como “BUENO” y 25% como “MUY BUENO” como se indica en la Figura 3.54.

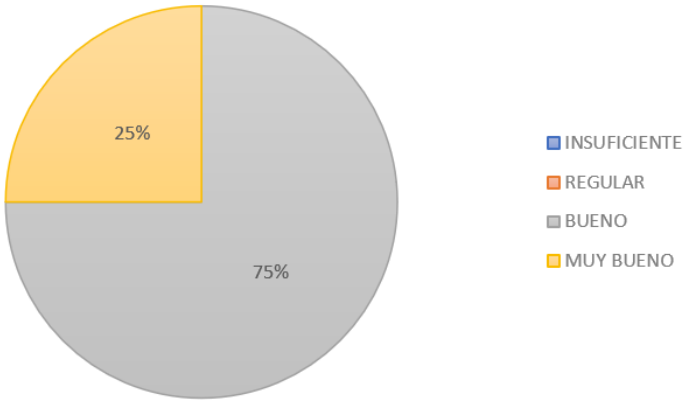


Figura 3.54. Respuesta a la decimotercera pregunta a Familiar

3.2.3. ROL PRINCIPAL

La primera pregunta busca comprobar si es posible crear el tipo de usuario. Los resultados indican que el 100% de los encuestados pueden crear el tipo de usuario como se puede ver en la Figura 3.55.

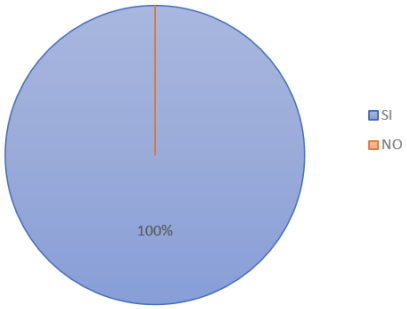


Figura 3.55. Respuesta a la primera pregunta a Principal

La segunda pregunta busca comprobar si al momento de crear su usuario pudo confirmarlo a través de un mensaje a su correo personal. Los resultados indican que el 100% de los encuestados reciben el correo para confirmarlo como se observa en la Figura 3.56.

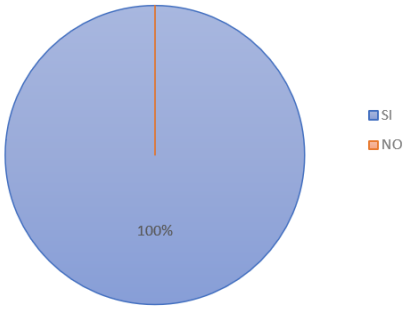


Figura 3.56. Respuesta a la segunda pregunta a Principal

La tercera pregunta busca comprobar si es posible entrar al menú de su rol. Los resultados indican que el 100% de los encuestados pueden ingresar al menú como se indica en la Figura 3.57.

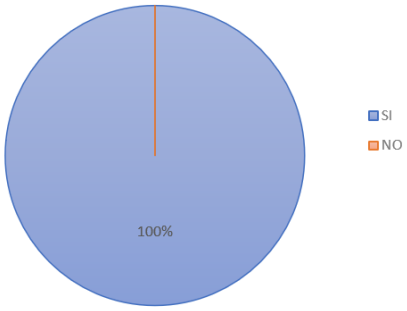


Figura 3.57. Respuesta a la tercera pregunta a Principal

La cuarta pregunta busca comprobar si al momento de cerrar su aplicación y volver a ingresar se le solicitó nuevamente la clave. Los resultados indican que el 100% de los encuestados no ingresan nuevamente la clave, sino que ingresa directamente a la interfaz de su rol como se muestra en la Figura 3.58.

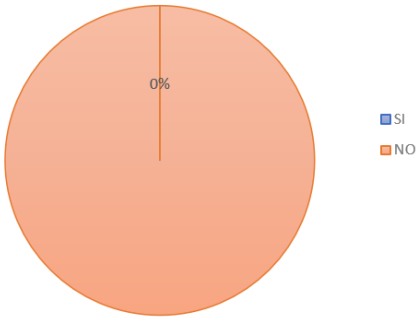


Figura 3.58. Respuesta a la cuarta pregunta a Principal

La quinta pregunta busca comprobar si pudo iniciar su transcripción y la comprendió de manera fácil. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se observa en la Figura 3.59.

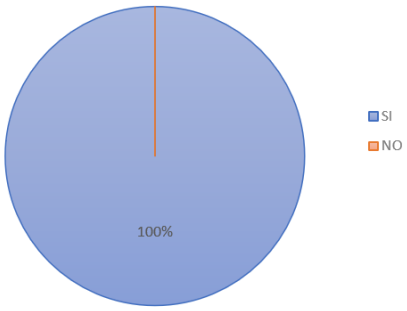


Figura 3.59. Respuesta a la quinta pregunta a Principal

La sexta pregunta busca comprobar si es posible pausar o continuar su transcripción. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se indica en la Figura 3.60.

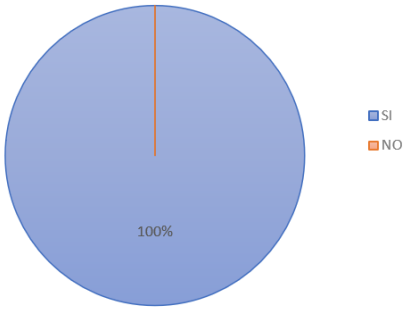


Figura 3.60. Respuesta a la sexta pregunta a Principal

La séptima pregunta busca comprobar si es posible aumentar o disminuir el tamaño de letra en el resultado de su transcripción. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se puede ver en la Figura 3.61.

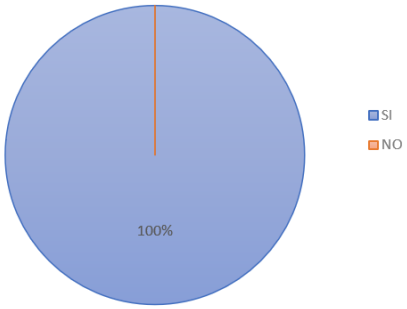


Figura 3.61. Respuesta a la séptima pregunta a Principal

La octava pregunta busca comprobar si es posible guardar su transcripción con el título que usted desee. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se muestra en la Figura 3.62.

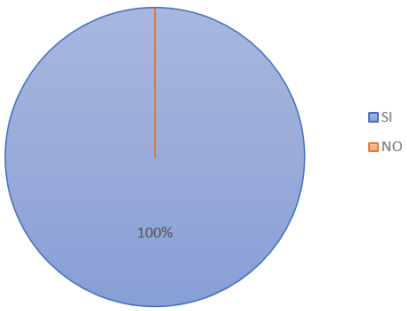


Figura 3.62. Respuesta a la octava pregunta a Principal

La novena pregunta busca comprobar si pudo observar sus transcripciones. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se indica en la Figura 3.63.

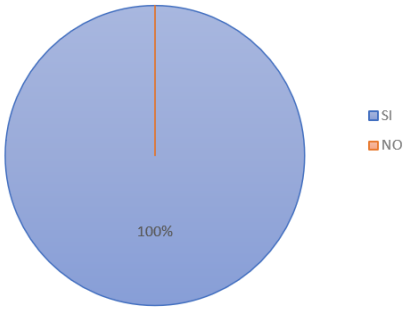


Figura 3.63. Respuesta a la novena pregunta a Principal

La décima pregunta busca comprobar si es posible editar cada una de sus transcripciones o eliminarlas. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se observa en la Figura 3.64.

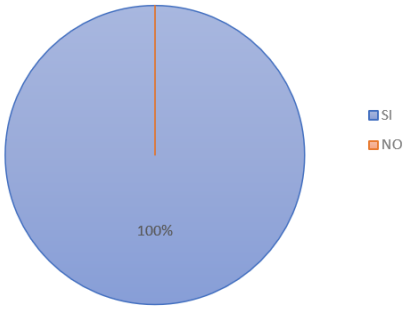


Figura 3.64. Respuesta a la décima pregunta a Principal

La decimoprimera pregunta busca comprobar si pudo observar un reporte de sus transcripciones por fecha. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se puede ver en la Figura 3.65.

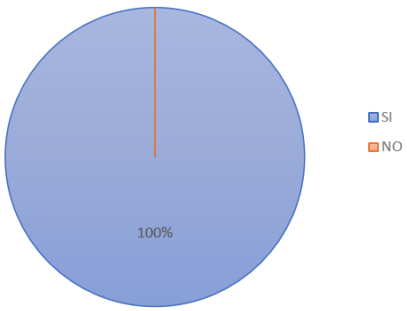


Figura 3.65. Respuesta a la decimoprimera pregunta a Principal

La decimosegunda pregunta busca comprobar si es posible descargar el reporte de sus transcripciones en formato PDF. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se muestra en la Figura 3.66.

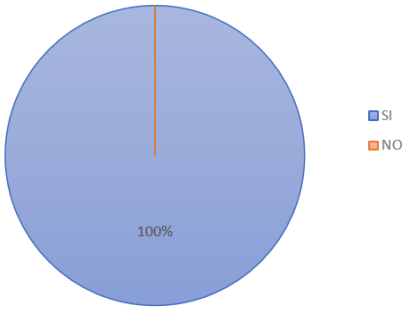


Figura 3.66. Respuesta a la decimosegunda pregunta a Principal

La decimotercera pregunta busca comprobar si se puede sincronizarse de manera fácil con su usuario familiar. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se observa en la Figura 3.67.

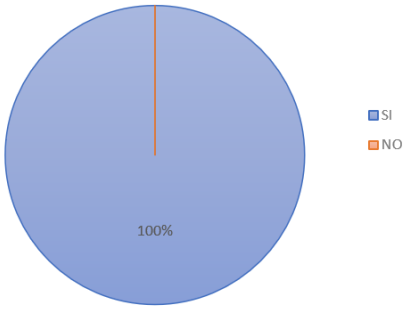


Figura 3.67. Respuesta a la decimotercera pregunta a Principal

La decimocuarta pregunta busca comprobar si se puede eliminar la sincronización con su usuario familiar de manera fácil. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se indica en la Figura 3.68.

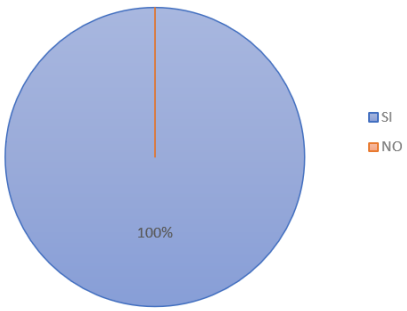


Figura 3.68. Respuesta a la decimocuarta pregunta a Principal

La decimoquinta pregunta busca comprobar si es posible ver su ubicación actual sin utilizar ningún otro aplicativo como Google Maps. Los resultados indican que el 100% de los encuestados pueden realizar esta función como se puede ver en la Figura 3.69.

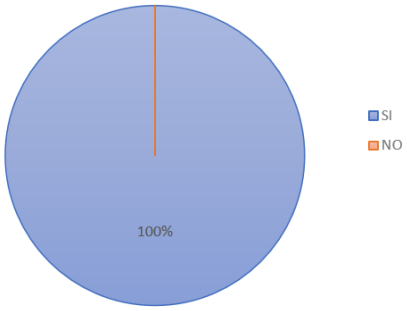


Figura 3.69. Respuesta a la decimoquinta pregunta a Principal

La decimosexta pregunta busca comprobar si se pudo enviar la ubicación a un contacto de su WhatsApp (si cuenta con la aplicación WhatsApp). Los resultados indican que el 100% de los encuestados pueden realizar esta función como se muestra en la Figura 3.70.

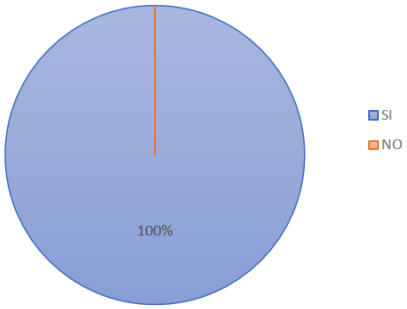


Figura 3.70. Respuesta a la decimosexta pregunta a Principal

La decimoséptima pregunta busca comprobar si es posible chatear con su usuario principal sincronizado. Los resultados indican que el 100% de los encuestados pueden chatear como se observa en la Figura 3.71.

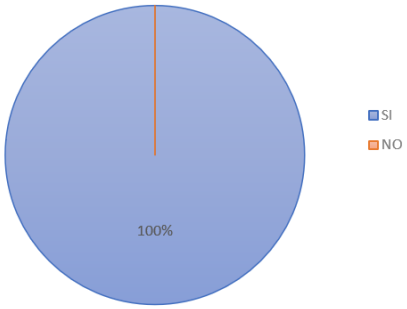


Figura 3.71. Respuesta a la decimoséptima pregunta a Principal

La decimoctava pregunta busca comprobar si encontró algún tipo de confirmación al momento de querer salir de la aplicación. Los resultados indican que el 100% de los encuestados recibieron la confirmación como se indica en la Figura 3.72.

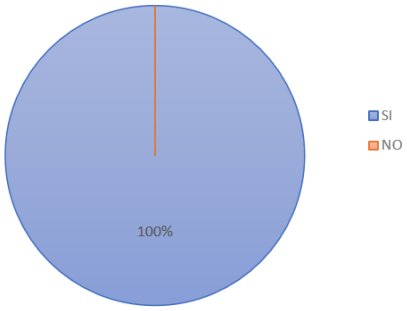


Figura 3.72. Respuesta a la decimoctava pregunta a Principal

La decimonovena pregunta busca comprobar si es posible regresar a cualquier parte que desee utilizando el menú desplegable. Los resultados indican que el 100% de los encuestados pueden dirigirse a cualquier parte del menú desplegable como se puede ver en la Figura 3.73.

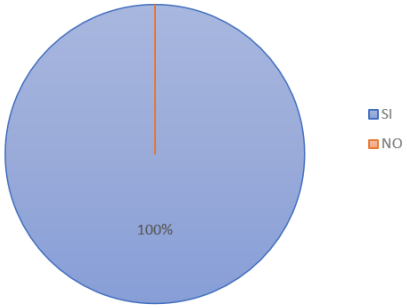


Figura 3.73. Respuesta a la decimonovena pregunta a Principal

La vigésima pregunta busca comprobar si es posible cerrar sesión. Los resultados indican que el 100% de los encuestados pueden cerrar sesión como se muestra en la Figura 3.74.

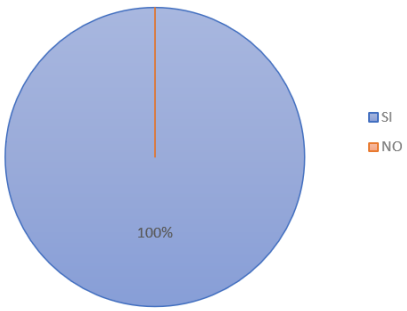


Figura 3.74. Respuesta a la vigésima pregunta a Principal

La vigesimoprimera pregunta busca comprobar si es posible que las personas con problemas auditivos puedan realizar sus transcripciones utilizando esta aplicación.

Los resultados indican que el 100% de los encuestados toman esta pregunta como “MUY BUENO” como se observa en la Figura 3.75.

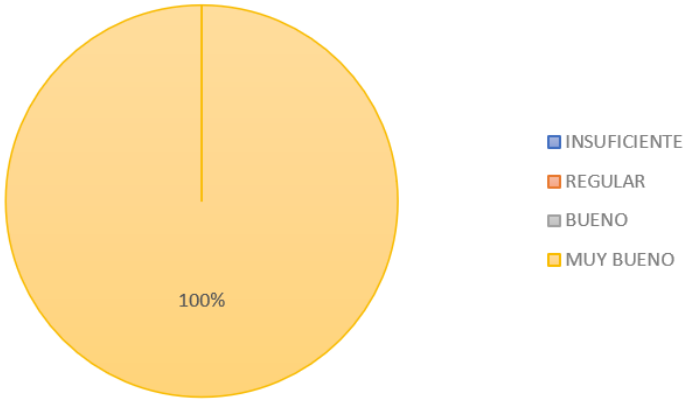


Figura 3.75. Respuesta a la vigesimoprimera pregunta a Principal

La vigesimosegunda pregunta busca comprobar qué le pareció el manejo de la aplicación respecto a velocidad. Los resultados indican que el 100% de los encuestados toman esta pregunta como “MUY BUENO” como se indica en la Figura 3.76.

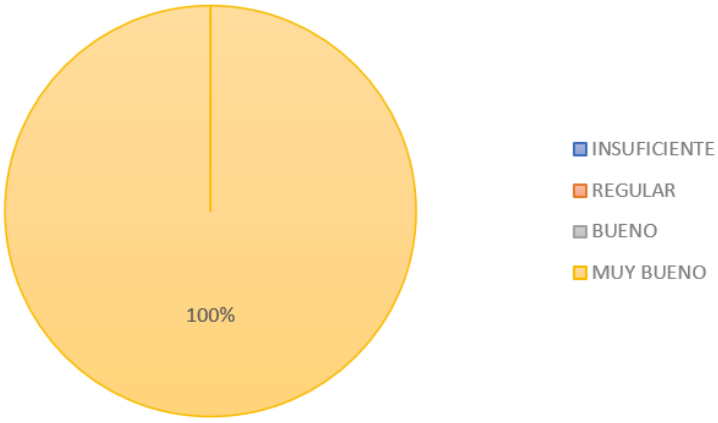


Figura 3.76. Respuesta a la vigesimosegunda pregunta a Principal

La vigesimotercera pregunta busca comprobar el manejo de sus transcripciones. Los resultados indican que el 75% de los encuestados toman esta pregunta como “BUENO” y 25% como “MUY BUENO” como se puede ver en la Figura 3.77.

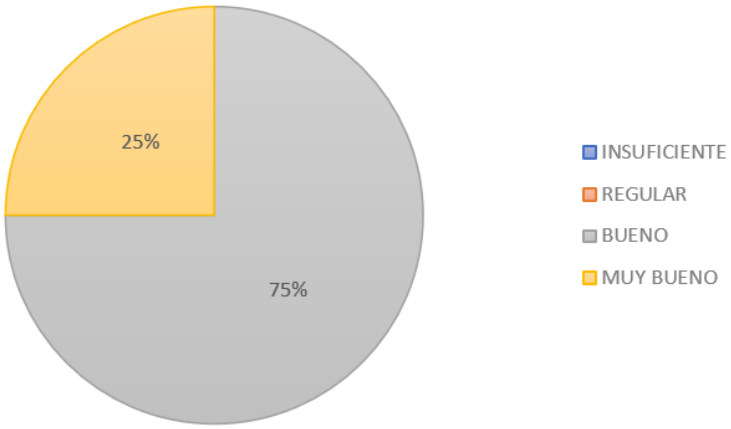


Figura 3.77. Respuesta a la vigesimotercera pregunta a Principal

La vigesimocuarta pregunta busca comprobar qué le pareció la parte gráfica (dibujos, letras, imágenes, colores). Los resultados indican que el 50% de los encuestados toman esta pregunta como “BUENO” y 50% como “MUY BUENO” como se muestra en la Figura 3.78.

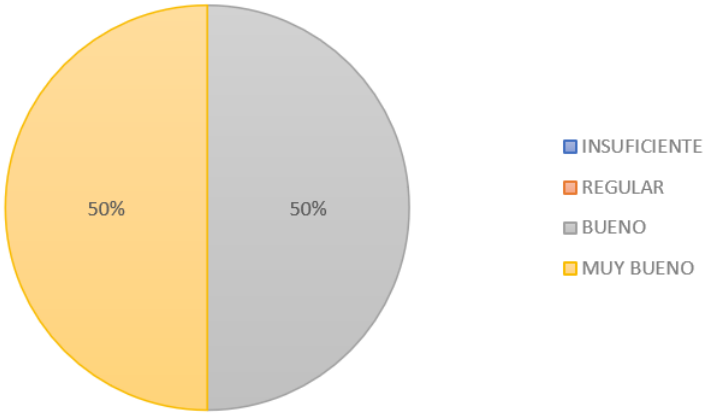


Figura 3.78. Respuesta a la vigesimocuarta pregunta a Principal

3.3. CORRECCIONES

Gracias a las diferentes pruebas de validación se encontraron errores los cuales fueron corregidos dando lugar a los resultados de la encuesta mostrada anteriormente; los errores corregidos se detallan a continuación:

- Los usuarios del rol Principal al momento de guardar una transcripción se quedaban en un ciclo infinito que no permitía continuar utilizando la aplicación, esto se solucionó cortando con procesos de caché propios de la memoria del dispositivo.
- Los usuarios del rol Familiar tenían un error al momento de ingresar al mapa para ver las diferentes ubicaciones del usuario Principal sincronizado, este problema se

debía a que el usuario Principal aun no tenía registros de ubicación; al momento si no existe puntos de ubicación el usuario Principal ya no cerrará la aplicación.

- Existía un error que no era visible en la aplicación pero que al crear varios usuarios se pudo detectar, este problema era que cuando el usuario olvidaba su clave y la cambiaba, esta acción no se veía reflejada en la base de datos, es decir, no se guardaba la nueva clave; se codificó que al momento de ingresar por primera vez al menú de cualquier rol se guarde la nueva clave.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El componente interactivo o aplicación y la base de datos son las bases fundamentales para este Trabajo de Titulación, ya que al trabajar de manera adecuada y sincronizada entre la aplicación y la base de datos se llega a la obtención de un aplicativo funcional y de fácil entendimiento para el usuario final.
- El componente interactivo o aplicación consta de tres tipos de roles (Principal, Familiar y Administrador) en donde se presenta un menú diferente para cada rol; se codificó de esta forma ya que cada rol debe cumplir con funciones propias que permiten una interacción adecuada entre el usuario y el aplicativo.
- Se utilizó una Activity global para recuperar contraseña o para cambiarla ya que se utiliza el mismo proceso en *Authentication* de *Firebase* para los dos procesos, por tal motivo, se generó una sola interfaz para este proceso permitiendo un ahorro en memoria interna de cualquier dispositivo al momento de instalación la aplicación.
- Al momento de generar el PDF tanto para el rol Principal como el Familiar se utilizó una Activity independiente (interfaz independiente dentro del programa que contiene programación), es decir no se utilizó fragment (interfaz reemplazable dentro de una Activity); se codificó de esta manera para evitar bucles con la base de datos y para evitar tiempos de espera prolongados entre la base de datos y la aplicación.
- Para realizar la autenticación del usuario en la aplicación se utilizó primero el método *getCurrentUser()* de *Authentication* en *Firebase* el cual permite comprobar que el usuario existe dentro de la base de datos, como segundo paso se verificó que el correo del usuario ya haya sido confirmado por el usuario a través de su correo personal utilizando el método *isEmailVerified()* de *Firebase;* para finalizar se busca el usuario en cada rol y se muestra el menú correspondiente.

- Para realizar la transcripción de voz a texto se utilizó android.speech.RecognizerIntent que extiende de java.lang.Object la cual pertenece a Android Studio; se codificó de esta manera para ahorrar recursos económicos ya que al utilizar este proceso la transcripción se vuelve total mente gratis con el único requisito de tener con conexión a internet, además android.speech.RecognizerIntent permite iniciar, pausar o continuar la transcripción sin necesidad de crear tres botones lo que implica ahorrar tiempo de ejecución y de programación.
- Al crear el PDF se debe cargar en el proyecto la librería *droidtext.0.4.jar* la cual es una librería para el diseño de documentos en PDF, para esto es necesario guardar la mencionada librería en Project/app/libs dentro de Android Studio, también configurar minSdkVersion en 16, targetSdkVersion en 29 y versionCode en 1; es necesaria esta librería para el documento en PDF ya que tiene varios atributos que permiten dar forma a los párrafos, color y tamaños a letras, centrar títulos, entre otras características y así tener un PDF entendible y al gusto del programador.
- Para crear los menús desplegables es necesario en el archivo androidManifest.xml que se encuentra en la dirección app/manifests dentro del programa, actualizar el tema a Theme.AppCompat.Light.NoActionBar, al no hacer esta parte la aplicación tendrá errores de ejecución que no se los puede encontrar ni con un analizador de código.
- La seguridad en la base de datos se encuentra segura de raíz por medio de *Firebase* a tal punto que se puede permitir o bloquear acceso a la información de diferentes nodos, es decir se puede configurar que un usuario solo tenga permisos de lectura, escritura o los dos respecto a un nodo de la base de datos; esto a nivel de firewall se conoce como “lista blanca” o “lista negra” de acceso.
- Los resultados de las encuestas indican que el prototipo es funcional y puede escalar a gran medida, por problemas de pandemia no fue posible trabajar en conjunto con el Consejo Nacional para Igualdad de Discapacidades (CONADIS), pero existe el interés para la creación de en un prototipo propio de ellos con las características fundamentales de este Trabajo de Titulación.
- Al momento de chatear entre los usuarios sincronizados, cada mensaje enviado será almacenado en la base de datos en tiempo real y se actualizará visualmente en la aplicación, esto se logró al llamar a la base de datos constantemente; si el receptor de un mensaje en el chat no está conectado a internet no le llegará el mensaje, pero cuando tenga conexión podrá visualizar todos los mensajes

inmediatamente; concluyendo que los tiempos de respuesta entre la aplicación y la base de datos al momento de chatear es de 5 a 7 milisegundos de respuesta.

- Para utilizar los servicios de *Google* y de *Firebase* se llamó a los repositorios como son *google()* y *jcenter(),* también se llamó a las dependencias *com.android.tools.build:gradle:3.6.4* y *com.google.gms:google-services:4.3.8* las cuales se ubican en *build.grade* a nivel de proyecto dentro del código de programación.

4.2. RECOMENDACIONES

- Para obtener datos de Database en Firebase existen dos maneras de hacerlo las cuales son *addValueEventListener* y *addListenerForSingleValueEvent,* por lo que se debe tomar en cuenta que utilizando addValueEventListener se estará siempre a escucha de cambios en la base de datos y se actualizará constantemente en la aplicación, mientras que addListenerForSingleValueEvent hará solamente un llamado a la base de datos, capturará estos datos y trabajará con ellos según la aplicación, es recomendable trabajar con addValueEventListener para chatear, en ubicaciones de tiempo real, pedidos, contador de movimiento o personas, entre otros.
- Para cada rol se generó una Activity independiente donde se cargó los diferentes fragments, por ejemplo, en el rol Principal se tiene seis fragments pertenecientes a sincronización, eliminar sincronización, transcripción, portafolio, reporte y chat; también incluye una Activity para ubicación y otra para generar el PDF; se debe codificar fragments separados y llamarlos cuando se desee ya que ayuda a que la aplicación sea fácil de utilizar e interactiva con el usuario.
- El prototipo puede ser modificado a través de líneas de código y configuración por la persona encargada del Proyecto de Titulación utilizando cualquier versión de Android Studio ya que se entregó el proyecto completo; esto se lo realizó para que la idea no se quede estancada y sea escalable y configurable para otra persona.
- Al momento de abrir el Proyecto de Titulación (código) en otra computadora es necesario actualizar la dirección de JDK en Android Studio y también sincronizar el archivo *build.gradle*; al no realizar estos pasos nunca se logrará visualizar las Activities y fragments completos dentro de Android Studio y se tendrá muchos errores.
- Es necesario que la clave de API no esté restringida en Google Cloud Platform para que el funcionamiento de Google Maps sea el correcto al momento de instalar la aplicación en un dispositivo.

- Es importante saber que Android Studio puede tener configuraciones que activen permisos en segundo plano, por ejemplo, se puede activar permisos de ubicación sin tener activado GPS en algunos celulares, por lo tanto, se recomienda no instalar aplicaciones descargadas directamente desde navegadores y utilizar antivirus.
- Es recomendable que al crear proyectos cada avance que se realice tenga varios respaldos en diferentes plataformas como puede ser Google, disco duro, flash, etc. Nunca confiarse de guardar la configuración temporalmente por un solo medio ya que en contacto con alguna actualización de Windows puede dañar el archivo base y por consecuencia eliminar todo el proyecto.
- Al trabajar en programación y creación de PDFs es posible que, aunque esté muy bien diseñado las líneas de programación y la aplicación indique que se descargó el PDF este archivo no se lo encuentre, por lo tanto, recomiendo instalar un buscador de archivos independiente al que viene por defecto en cada dispositivo.
- Al momento de guardar un archivo PDF desde una aplicación creada en Android Studio el archivo descargado no puede ser visible en algunos dispositivos, es por eso que es recomendable instalar una aplicación de terceros desde la Play Store que permita visualizar los archivos descargados en el dispositivo sin restricciones.
- Este prototipo no funciona correctamente en celulares que se haya bloqueado o tenga restricciones con plataformas de Google (algunos equipos HUAWEI), se recomienda buscar un simulador de aplicaciones en el dispositivo o un aplicativo para el correcto funcionamiento del prototipo.
- Sería recomendable que este prototipo pueda ser analizado en asignaturas de programación, puesto que, por los resultados de la evaluación existe una gran acogida a la idea no solo por el tema de transcripción sino por la confianza de guardar información y obtenerla en tiempo real.
- El prototipo está diseñado netamente para dispositivos Android, para futuros desarrollos se lo puede realizar en una plataforma web o más aun en plataforma web online, con la recomendación de un nuevo diseño web.
- En un futuro, se podría generar un apartado dentro de cada menú donde se obtenga información de cómo utilizar la aplicación dado que esto no fue pensado en un inicio en el plan de trabajo propuesto.
- Como alternativa para mejorar la aplicación a gran escala, se puede generar filtros en varias partes de los diferentes menús, una recomendación puede ser que cuando se visualiza las transcripciones se pueda incluir un filtro que busque una

transcripción por su tema ya que al momento se debe desplazar hasta encontrar la transcripción deseada.

- Dado que la industria referente a la información en tiempo real está creciendo aceleradamente y a escala mundial, resulta muy importante enseñar y motivar a los estudiantes en el desarrollo de aplicaciones, ya que es una ventaja que genera recursos monetarios.

BIBLIOGRAFÍA

[1] Developers, «Android plataform. Developers Android,» 24 Febrero 2021. [En línea]. Available: https://developer.android.com/reference/android/speech/RecognizerIntent.

[2] Menéndez, Android 100%, edición 1, España: Creative Commons, 2014.

[3] GOOGLE, «Firebase,» 11 Abril 2021. [En línea]. Available: https://firebase.google.com/.

[4] A. Peralta, «Metodología SCRUM,» Universidad ORT Uruguay, Uruguay, 2003.

[5] G. L. J. P. Alexander Menzinsky, Scrum Manager, Madrid: Safe Creative, 2016.

[6] Android, «Que es Android,» Android, [En línea]. Available: https://www.android.com/intl/es_es/what-is-android/. [Último acceso: 17 Abril 2021].

[7] A. S.O., «Android 11,» Android, [En línea]. Available: https://www.android.com/android-11/. [Último acceso: 17 Abril 2021].

[8] Android, «Android 6.0 Marshmallow,» Android, [En línea]. Available: https://www.android.com/intl/es_es/versions/marshmallow-6-0/. [Último acceso: 17 Abril 2021].

[9] Android, «Android 9,» Android, [En línea]. Available: https://www.android.com/versions/pie-9-0/#adaptive-technologies. [Último acceso: 17 Abril 2021].

[10] S. d. Comunicaciones, «Arquitectura Android,» [En línea]. Available: https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android. [Último acceso: 17 Abril 2021].

[11] Developers, «Developers,» 9 Febrero 2021. [En línea]. Available: https://developer.android.com/studio/intro?hl=es-419.

[12] Á. d. p. y. Desarrollo, «Curso de Introducción a Java,» [En línea]. Available: https://www.mundojava.net/caracteristicas-del-lenguaje.html?Pg=java_inicial_4_1.html. [Último acceso: 17 Abril 2021].

[13] I. Firebase, «Firebase Inicial,» [En línea]. Available: https://firebase.google.com/?hl=es. [Último acceso: 17 Abril 2021].

[14] A. Dufeter, «Alex Dufeter,» 3 Octubre 2017. [En línea]. Available: https://firebase.googleblog.com/2017/10/introducing-cloud-firestore.html.

[15] Firebase, «Firebase,» 15 Abril 2021. [En línea]. Available: https://firebase.google.com/products/auth?gclid=Cj0KCQjwyN-DBhCDARIsAFOELTlBydTNn00Dfo-reVghnMvti2daGsVCL1pYItp70s7bDEgz35e-vXgaApgLEALw_wcB&gclsrc=aw.ds.

[16] GooglePlayServices, «GooglePlayServices,» [En línea]. Available: https://developers.google.com/android?hl=es. [Último acceso: 17 Abril 2021].

[17] Google Developers, «Maps SDK for Android,» [En línea]. Available: https://developers.google.com/maps/documentation/android-sdk/overview?hl=es. [Último acceso: 17 Abril 2021].

[18] Google Developers, «Geocoding API,» [En línea]. Available: https://developers.google.com/maps/documentation/geocoding/start?hl=es. [Último acceso: 17 Abril 2021].

[19] Google Developers, «Geolocation API,» [En línea]. Available: https://developers.google.com/maps/documentation/geolocation/overview?hl=es. [Último acceso: 17 Abril 2021].

[20] I. Sommerville, Ingeniería del software, Madrid: Pearson Educación, S. A., 2005.

[21] D. Ramos, «Desarrollo de Software: Requisitos, estimaciones y análisis,» [En línea]. Available: https://books.google.com.ec/books?id=tBaYCwAAQBAJ&printsec=frontcover&dq=qu#v=onepage&q&f=false. [Último acceso: 19 Abril 2021].

[22] j. Abad, Historias de usuario, una visión pragmática, Copyrigth, 2018.

[23] C. J. d. Parga, UML, Aplicaciones en JAVA y C++, Madrid: RA-MA, S.A. Editorial y Publicaciones, 2015.

[24] D. Gutierrez, «UML Diagrama de Clases,» Universidad de los Andes, Mérida, 2011.

[25] J. R. L. J. D. C. J. Casas Anguita, La encuesta como técnica de investigación., Madríd: Departamento de Planificación y Economía de la Salud, 2002.

[26] Organización Mundial de la Salud, «Sordera y pérdida de audición,» 15 Marzo 2019. [En línea]. Available: https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss.

[27] C. Jiménez, UML Aplicaciones en Java y C++, Madrid: RA-MA, S.A., 2015.

ANEXOS

ANEXO A. Proyecto (comprimido)

ANEXO B. Guía de usuario administrador

ANEXO C. Guía de usuario familiar

ANEXO D. Guía de usuario principal

ANEXO E. Encuestas

ANEXO F. Guía activar proyecto en otra PC